HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Carsten Zahn

Integration einer BPM-Engine in eine komplexe Geschäftsanwendung

2013

Fakultät:
Mathematik/Naturwissenschaften/Informatik

BACHELORARBEIT

Integration einer BPM-Engine in eine komplexe Geschäftsanwendung

Autor/in:
Herr Carsten Zahn

Studiengang:
Wirtschaftsinformatik

Seminargruppe:
WF10w1-B

Erstprüfer:
Prof. Dr.-Ing. Wilfried Schubert

Zweitprüfer:
Dipl.-Ing. (BA) Jan Scherf

Einreichung:
Mittweida, 11.10.2013

Faculty of
Mathematics/Sciences/Computer Science

BACHELOR THESIS

Integration of BPM Engine in a complex Business Application

author:
Mr. Carsten Zahn

course of studies:
Wirtschaftsinformatik

seminar group:
WF10w1-B

first examiner:
Prof. Dr.-Ing. Wilfried Schubert

second examiner:
Dipl.-Ing. (BA) Jan Scherf

submission:
Mittweida, 11th October 2013

Bibliografische Angaben

Zahn, Carsten:
Integration einer BPM-Engine in eine komplexe Geschäftsanwendung
Integration of BPM Engine in a complex business application – 2013
84 Seiten gesamt, 59 Seiten Inhalt, 16 Seiten Anhang, 35 Abbildungen, 4 Tabellen,
Mittweida, Hochschule Mittweida - University of Applied Sciences,
Fakultät: Mathematik/Naturwissenschaften/Informatik, Bachelorarbeit, 2013

Referat

Geschäftsprozessmanagement, auch Business Process Management, ist ein Trend, welcher in der Wirtschaft kaum noch wegzudenken ist. Allerdings hat sich dieser Trend nicht nur in der Wirtschaft entwickelt, sondern auch im IT-Bereich. Dort wurde in den letzten Jahren vermehrt auf geschäftsprozessunterstützender Software, sogenannte Business Process Management Systeme, gesetzt.

Die vorliegende Arbeit befasst sich mit der Integration solch eines Business Process Management Systems, speziell dessen Business Process Management Engine, in eine komplexe Geschäftsanwendung. Dabei werden Schwerpunkte für die Integration anhand eines Beispiel Business Process Management Systems und einer Beispiel Geschäftsanwendung beschrieben und entsprechende Lösungen vorgestellt. Außerdem folgt ein Einblick in ausgewählte Funktionen eines Business Process Management System.

Inhaltsverzeichnis

Literaturverzeichnis

Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

„Wer die Prozesse im Unternehmen nicht beherrscht,
beherrscht das ganze Unternehmen nicht.“[1]

William Edwards Deming

Unternehmen haben das Ziel, die Effizienz und die Qualität ihrer Geschäftsprozesse kontinuierlich zu erhöhen. Geschäftsprozessmanagement, auch als Business Process Management (BPM) bezeichnet, ist ein Trend, welcher einen wichtigen Schlüsselfaktor darstellt.

Es werden Modelle erstellt, welche die Geschäftsprozesse abbilden. Bei der Erstellung dieser Modelle ist es dem Mitarbeiter möglich, auf das standardisierte Business Process Modeling and Notation (BPMN) zurückgreifen. Auf Basis dieser Standardisierung kann jeder Mitarbeiter in der Lage sein, das Modell eines Geschäftsprozesses zu lesen, zu verstehen und Verbesserungen anzubringen. Gerade bei der Optimierung von Prozessen sind solche Modelle von Bedeutung. Durch die Analyse dieser Modelle zeigen sich Schwachpunkte im Prozess und mögliche Verbesserungen sind schneller erkennbar.

Ebenfalls hat sich im IT-Bereich in den letzten Jahren der Einsatz von geschäftsprozessunterstützender Software bewährt.[2] Rechnergestützte Prozesse werden durch Workflow Management Systeme (WfMS) oder BPM-Systeme (BPMS) unterstützt. Der Reifegrad der verfügbaren Systeme hat einen Stand erreicht, bei dem der Einsatz solcher BPMS lohnenswert ist. Vor allem in Unternehmen, die prozessorientiert arbeiten, ist die Integration eines derartigen Systems in die benutzte Geschäftsanwendung von Vorteil.

1.1 Problembeschreibung und Zielsetzung

Für Softwareentwicklungsunternehmen wie die DELTA proveris AG, lohnt sich die Integration eines BPMS in die programmierte Geschäftsanwendung, um den Wert des eigenen Produktes zu steigern und es für Kunden attraktiver zu gestalten.

[1] [DEM13] (Deming, 12.09.2013 – 11:24)
[2] [FUN10] (Funk, Gómez , Niemeyer, & Teuteberg, 2010, S. 7) &
[HOH11] (Hohwiller & Dr. Schlegel, 2011, S. 80)

Des Weiteren haben die Softwareentwickler eine bessere Kontrolle über den Quelltext, da der Prozess nicht mehr programmiert, sondern über ein BPMN-Modell abgebildet wird. Die leicht zu vollziehende Änderung des Prozessdiagramms führt dazu, dass der Softwareentwickler sich auf die Logik der Prozesse konzentrieren kann.
Für die Kunden ist es einfacher die Änderungen ihrer Geschäftsprozesse zu übermitteln. Sie entwerfen dazu ein BPMN-Modell mit den Änderungen, welches der Softwareentwickler 1:1 in die Geschäftsanwendung übernimmt und mit der alten oder neuen Logik anreichert.

Die DELTA proveris AG beabsichtigt ebenfalls solch eine Lösung in ihre Geschäftsanwendung Leasman zu integrieren. Leasman ist ein betriebswirtschaftliches „[….] Management-Komplettsystem, für den Leasing- und Finanzierungsbereich [….].“[3] Neben den Leasingbereich deckt Leasman auch Funktionen der allgemeinen Fuhrparkverwaltung ab. Leasman besteht aus mehreren Komponenten, welche die verschiedenen Funktionalitäten kapseln (siehe Abbildung 1 – Business Logic Server). Die steigende Nachfrage von mehreren Interessenten für derartige Softwarelösungen im Leasingbereich führt ebenfalls zum Wunsch der Integration eines BPMS in die Leasman-Systemlandschaft.

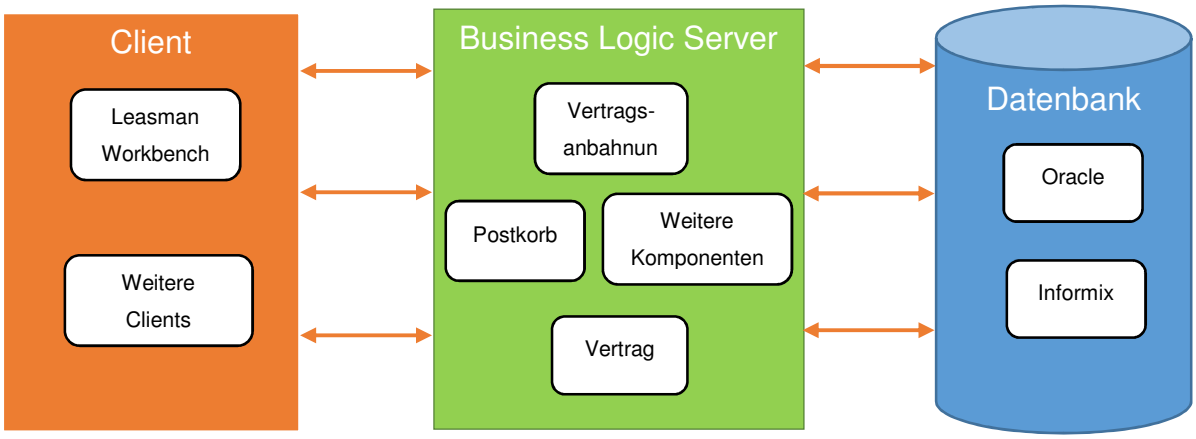


Abbildung 1: Die Leasman-Systemlandschaft

Momentan sind die rechnergestützten Prozesse der Kunden fest im Quelltext hinterlegt. Änderungen innerhalb dieser Prozesse müssen deshalb durch die Entwicklungsabteilung der DELTA proveris AG erfolgen. Dies ist zeitaufwändig und verlangt eine reibungslose Kommunikation zwischen Unternehmen und Kunden.

[3] [DEL13] (DELTA proveris AG - Leasman, 2013)

Durch den Einsatz einer BPM-Engine ist es möglich, einen Teil der Aufgabe „Änderung des Kundenprozesses“ an den Kunden zu übertragen. Der Kunde modelliert seine Änderungen selbst, zum Beispiel mit Hilfe eines BPMN-Editors. Ein Mitarbeiter der DELTA proveris AG erstellt anschließend das BPMN-Modell für Leasman und pflegt dieses ein. Damit kann der Kunde den Geschäftsprozess seinen Wünschen entsprechend anpassen.

Ziel dieser Arbeit ist es zu klären, wie sich eine solche BPM-Engine in Leasman integrieren lässt und ob die damit auftretende Komplexität beherrschbar bleibt. Am Beispiel der BPM-Engine Activiti soll die geforderte Integration erfolgen. Die dabei entstehenden Herausforderungen müssen erörtert, diskutiert und gegenüber den aufgestellten Anforderungen validiert werden.

Für die Integration einer BPM-Engine und zur Erklärung einiger Funktionen wird der bereits bestehende Prozess „Vertragsanbahnung“ von Leasman genutzt. Die Realisierung der Integration zeigt einen praktischen Nachweis für dessen Machbarkeit mittels eines Prototyps.

Auf Basis dieser Aufgabenstellung lassen sich folgende Fragen formulieren:

- Welche Voraussetzungen bringt eine BPM-Engine mit, um als Embedded-Lösung eingesetzt zu werden?
- Welche Anforderungen stellt Leasman, damit eine Applikation integriert werden kann?
- Ist eine Anpassung der BPM-Engine an die Funktionen des Leasman nötig und wie findet dies gegebenenfalls statt?

1.2 Aufbau und Abgrenzungen

Das folgende Kapitel startet mit einem kurzen Einblick in das BPM und das BPMS.

Anschließend wird im dritten Kapitel eine Engine, in dem Fall Activiti, beschrieben und analysiert. Die Hauptbetrachtung liegt hierbei beim Aufbau und dem Einsatz der Engine. Kapitel vier zeigt anhand des Leasman-Systems, worauf bei der Analyse einer Geschäftsanwendung zu achten ist. Diese beiden Kapitel dienen als Grundlage für das zu erstellende Integrationskonzept.

Das fünfte Kapitel widmet sich der Erstellung des Integrationskonzepts. Zunächst werden die Anforderungen an diese Herausforderung erörtert und anschließend ein Lösungsmodell erarbeitet.

Die Realisierung dieses Konzeptes sowie die Diskussion der aufgetretenen Probleme und Lösungsstrategien erfolgt im sechsten Kapitel.

Das Abschlusskapitel zieht ein Fazit, in wie weit die Ziele dieser Arbeit erfüllt wurden. Ein Ausblick schließt diese Arbeit ab.

2. Grundlagen

2.1 Business Process Management

Der Begriff Business Process Management besitzt keine anerkannte Standarddefinition, sondern wird von vielen Autoren und BPM-Experten unterschiedlich definiert, erweitert oder verändert. Prof. Gadatsch definiert den Begriff BPM wie folgt:

> „Prozess Management ist ein zentraler Bestandteil eines integrierten Konzeptes für das Geschäftsprozess- und Workflow Management. Es dient dem Abgleich mit der Unternehmensstrategie, der organisatorischen Gestaltung von Prozessen sowie deren technischer Umsetzung mit geeigneten Kommunikations- und Informationssystemen.“[4]

Die Gartner Inc. erweitert den Begriff in Bezug auf die Kunden:

> „[…] linking together people, information flows, systems and other assets to create and deliver value to customers and constituents.“[5]

Zusammengefasst bedeutet BPM:

> *eine zielstrebige, inhaltlich abgeschlossene und eine zeitlich korrekte Anordnung von Aktionen (auch Aktivitäten und Funktionen), welche von mehreren Organisationen, Gruppen (Groups) oder Benutzern (User) verwaltet oder bearbeitet werden, damit eine unternehmensspezifische Anforderung (oder Ziel) erfüllt wird.*[6]

Diese Schwerpunkte führen zu einer bestimmten Frage, dessen Basis es ermöglicht die Ziele von BPM näher zu bestimmen und zu betrachten.

> „Wer (Akteure) macht was (Aufgaben), wann (zeitliche Abfolge), wie (Qualität), womit (Ressourcen) und zu welchem Zweck (Unternehmensziele)?“[7].

---

[4] [GAD10] (Gadatsch, 2010], S.1)
[5] [GAR13] (Gartner Inc. - BPM, 2013)
Übersetzung: […] Verknüpfung von Personen, Informationsfluss, Systeme und anderen Vermögen, zur Erstellung und Lieferung eines Mehrwertes für Kunden und Auftraggeber [eigene Übersetzung]

[6] [MÜL11] (Müller J. , 2011, S. 8 f)
[7] [FUN10] (Funk, Gómez , Niemeyer, & Teuteberg, 2010, S.9 f)

Zu den Zielen des BPM gehören unter anderem die Überwachung, die Dokumentation, die Steuerung, die Identifizierung, die Planung, Gestaltung und Modellierung sowie eine kontinuierliche Verbesserung des Unternehmensprozesses.

Bei deren Umsetzung wird zwischen zwei Sichten unterscheiden. Eine klassische Sicht als Berater für die Analyse und Optimierung von Geschäftsprozessen ist möglich, bis hin „zur technischen Integration einer prozessgesteuerten Softwarekomponente bei der Entwicklung einer Fachanwendung“[8].

2.2 Geschäftsprozess oder Workflow

Neben dem Begriff BPM gibt es noch den Begriff Workflow Management (WfM). Die Bezeichnungen für deren Prozesse ähneln sich inhaltlich, weswegen häufig eine Verwechslung auftritt. Um zu klären, ob es sich um ein Business Process (Geschäftsprozess) oder einen Workflow handelt, sind die Definitionen von Prof. Gadatsch [GAD10] hilfreich. Er fasst, aus mehreren bereits vorhandenen Definitionen, eine Definition zusammen.

Business Process / Geschäftsprozess

> *„Ein Geschäftsprozess ist eine zielgerichtete, zeitlich logische Abfolge von Aufgaben, die arbeitsteilig von mehreren Organisationen oder Organisationseinheiten unter Nutzung von Informations- und Kommunikationstechnologien ausgeführt werden können. Er dient der Erstellung von Leistungen entsprechend den vorgegebenen, aus der Unternehmensstrategie abgeleiteten Prozesszielen.“*[9]

Workflow

> *„Ein Workflow ist ein formal beschriebener, ganz oder teilweise automatisierter Geschäftsprozess. Er beinhaltet die zeitlichen, fachlichen und ressourcenbezogenen Spezifikationen, die für eine automatische Steuerung des Arbeitsablaufes auf der operativen Ebene erforderlich sind. Die hierbei anzustoßenden Arbeitsschritte sind zur Ausführung durch Mitarbeiter oder durch Anwendungsprogramme vorgesehen.“*[10]

[8] [HOH11] (Hohwiller & Dr. Schlegel, 2011, S. 80)
[9] [GAD10] (Gadatsch, 2010, S. 41)
[10] [GAD10] (Gadatsch, 2010, S. 47)

Innerhalb der Beschreibung einer BPM-Engine wird keine Unterscheidung zwischen Workflow und Geschäftsprozess getroffen, da sowohl WfM, als auch BPM genutzt werden. Trotzdem muss der Prozess-Analyst (siehe Kapitel 2.3) stets die Unterschiede zwischen den beiden Prozessbeschreibungen im Hinterkopf haben.

2.3 Rollen und Lebenszyklus von Business Process Management

Während der Nutzung von BPM interagieren unterschiedliche Personen in unterschiedlichen Phasen von der Erstellung (Design) bis zur Auswertung (Optimization) eines Geschäftsprozess (vgl. [FRE10] S.17). Dabei kann der Prozess

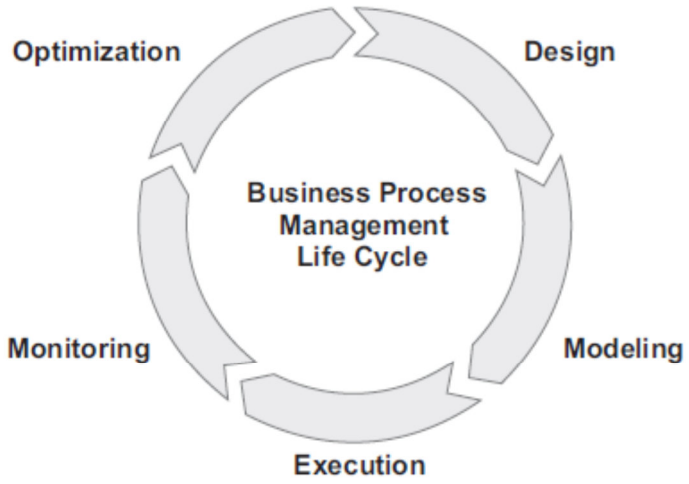


Abbildung 2: Phasen des BPM-Lebenszyklus[11]

mit und ohne BPMS erfolgen.

Design-Phase

Der **Prozess-Analyst** ist der Hauptakteur in dieser Phase. Er sammelt Informationen, wie einzelne Aufgaben, Nutzergruppen oder eventuell auftretende Probleme, analysiert und wertet diese aus. Auf Basis dieser Informationen fertigt er ein Prozessdiagramm an, welches den optimalen Soll-Prozess aufzeigt.

Modeling-Phase

Der Prozess-Analyst kann auch der **Prozess-Engineer** sein, der in dieser Phase das Prozessdiagramm bearbeitet. Bei der Verwendung eines BPMS wird das Diagramm importiert und die Aktivitäten mit Funktionen und Logik hinterlegt. Dadurch entsteht die Prozessdefinition. Der Prozess Engineer dokumentiert anschließend das Prozessdiagramm oder die Proessdefinition.

[11] [RAD12] (Rademakers, 2012, S. 21)

Execution-Phase

Nach erfolgreicher Validierung ist der Prozess startbar. Die Ausführung des Prozesses erfolgt in der Execution-Phase. Die **Prozessbeteiligten** (Mensch oder Maschine) führen dabei die Aktivitäten des Geschäftsprozesses aus. Rechnergestützte Aufgaben können in dieser Phase durch BPMS unterstützt werden.[12]

Monitoring-Phase

Während die einzelnen Aufgaben ausgeführt werden, kann der komplette Prozess überwacht werden. Unter anderen wird geprüft, ob die Unternehmensziele erreicht werden, nicht berücksichtigte Fehler auftreten und weiteres. Diese Informationen sind Ad-Hoc verfügbar und werden für die Optimierung des Geschäftsprozesses benötigt. Die Person, die für die Überwachung des Prozesses zuständig ist, ist der **Prozess-Administrator**. Er kann jederzeit in den Prozess eingreifen. BPMS helfen hierbei bei der Speicherung und Auswertung von Daten.

Optimization-Phase

In dieser Phase wird überprüft, in wie weit der Prozess optimiert und verbessert werden kann. Dazu wird ein Report an den **Prozess-Manager** geschickt. Der Report beinhaltet die Resultate des Monitorings und den Soll-Ist-Zustand des Prozesses. Mit diesen Resultaten und durch den Einsatz anderer Techniken oder Verbesserungen wird festgestellt, wie der Prozess optimiert werden kann. Anschließend erfolgt ein Feedback an den Prozess-Analyst, der den Prozess entsprechend der Auswertung überarbeitet.

2.4 Automatisierung mittels Business Process Management System

In den vorherigen Kapitel wurde immer wieder auf BPMS eingegangen und wie es die Ausführbarkeit und Optimierung von Prozessen verbessert. Die Definition von Prof. Komus beschreibt, was BPMS bedeutet:

> *BPMS ist ein „[…] System aufeinander abgestimmter (interoperabler), spezialisierter IT-Anwendungen, die ein breites Spektrum aller notwendigen Funktionen zur Modellierung und Simulation, Analyse, Entwicklung und zum Betrieb von BPM-Lösungen unterstützen […]“*[13].

[12] [RAD12] (Rademakers, 2012, S. 22)
[13] [KOM11] (Komus, 2011, S. 7)

Prof. Gadatsch erweitert den Begriff durch den Anwendungsort:

> *BPMS „[…] spiegeln Geschäftsprozesse in der Informationstechnik.“*[14]

Zusammengefasst stellt BPMS IT-unterstützende Mittel zur Verfügung, um den in Kapitel 2.3 beschriebenen Lebenszyklus abzubilden, zu automatisieren und zu optimieren. Dazu kann solch ein System eigenständig im Unternehmen eingesetzt oder in die Geschäftsanwendung integriert werden. Die Integration beschreibt die:

> *Verknüpfung verschiedener Anwendungen und Systemen, die zusammen arbeiten. Sie greifen auf ein einheitliches Datenmodell zu und sind semantisch gleich aufgebaut.*[15]

Durch den Zugriff des BPMS auf Komponenten der Geschäftsanwendung entstehen Kopplungen. Diese Kopplungen sind so gering wie möglich zu halten. Kopplung ist:

> *die Verknüpfung von Systemen, Softwaremodulen und Anwendungen. Außerdem beschreibt es ein Maß für die Abhängigkeiten untereinander.* [16]

Anhang D zeigt nochmals eine Abbildung über die Funktionalität eine BPMS. Literatur, wie „Activiti in Action“ [RAD12] oder „Workflow-based Integration“ [MUL05] geben einen Einblick in die Funktionen von BPM-Systemen.

2.5 Business Process Modeling and Notation

Durch den Fortschritt des BPM und den damit verbundenen Drang, Prozesse ausführbar zu machen, entwickelten sich immer mehr BPM-Sprachen (siehe Anhang B), die die Modellierung und Ausführbarkeit von Prozessen ermöglichen.

Diese Umsetzungen beinhalten solch weitreichende Ausprägungen, dass Anfangs die Standardisierung ein großes Problem darstellte. Die Object Management Group (OMG), „[…] ein Konsortium, das sich u.a. mit der herstellerunabhängigen Entwicklung von Standards beschäftigt.“[17], drängte immer mehr darauf, eine einheitliche Notation für das BPM zu entwickeln. Diese standardisierte Notation muss vor allem dafür Sorge tragen, dass der Umgang bei der Modellierung von Geschäftsprozessen mit Tools (Werkzeugen) ohne größere Probleme zügig abläuft und von jeder Person sofort gelesen und verstanden werden kann.

[14] [GAD10] (Gadatsch, 2010, S. 253)
[15] [WIN13] (Wikipedia - Integration, 09.09.2013 – 12:20)
[16] [WKO13] (Wikipedia - Kopplung, 09.09.2013 – 15:14)
[17] [GAD10] (Gadatsch, 2010, S. 97)

Im Jahr 2005 entwickelt IBM die BPMN, welche im Jahre 2005 von der OMG, übernommen und gepflegt wurde. Die OMG erklärte BPMN 2006 zum offiziellen Standard. Mit der BPMN 2.0 steht seit Anfang 2011 eine standardisierte Prozessnotation zur Verfügung, die nicht nur vielseitig einsetzbar, sondern auch mächtig an Funktionen ist.[18]

Die Nutzung dieser standardisierten Notation findet auch in BPMS Anwendung. So werden BPM-Editoren genutzt, um das Modell eines Geschäftsprozesses abzubilden und für die Engine in ein nutzbares Format umzuwandeln.

Geschäftsprozesse, deren Abbildung durch diese Werkzeuge lohnenswert ist, sollten gewisse Voraussetzungen erfüllen. Zunächst muss der Prozess automatisierbar sein. Das bedeutet, wenn der Prozess nur durch Menschenhand und ohne Einsatz von Rechnersystemen ausgeführt wird, ist das Nutzen solcher Systeme unnütz. Des Weiteren muss ein Mitarbeiter vor dem Einsatz eines BPMS recherchieren, welche Software für das Unternehmen geeignet ist. Welche Kriterien dabei eine Rolle spielen zeigt Kapitel 2.8. Auch eine Regelmäßigkeit der Prozesse muss vorhanden sein, da ein „Einmalprozess“ (vgl. [GAD10] S.253) Ressourcen, Personal und Zeit unnötig verbraucht.

Solch ein Prozess kann wie folgt aussehen:

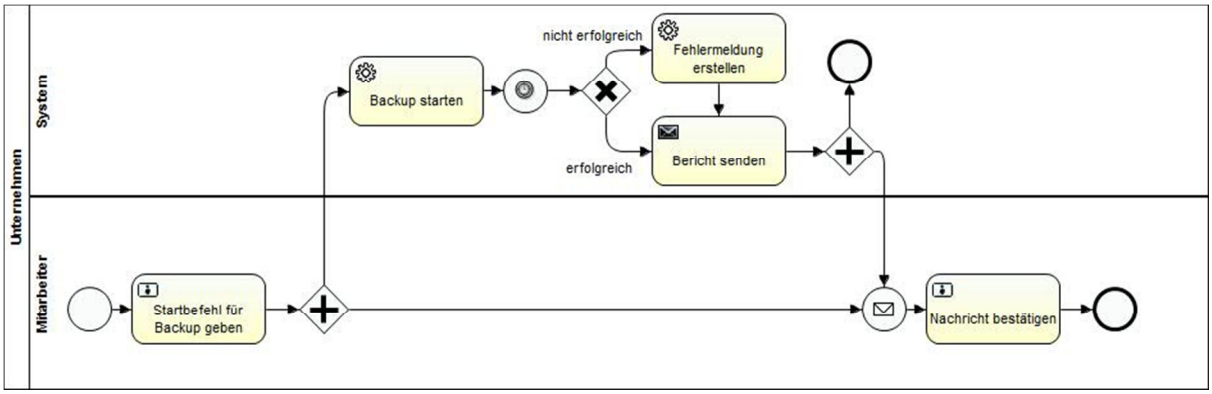


Abbildung 3: Beispielprozess - Backupvorgang[19]

[18] [FUN10] (Funk, Gómez , Niemeyer, & Teuteberg, 2010, S. 37)
[19] *erstellt über den Activiti-Designer

Der fiktive Prozess des Backupvorgangs zeigt relevante Elemente des BPMN. Zunächst wird das Element „Unternehmen“ untersucht. Dies ist das Pool-Element. Ein Pool stellt eine Organisationseinheit, welche aus einem oder mehreren Aufgabenträgern (Swimlanes) besteht, im Beispiel Mitarbeiter und System. Jede Lane beinhaltet die Elemente, die während des Prozesses von dem entsprechenden Aufgabenträger vollzogen werden.

Die Lane „Mitarbeiter“ beinhaltet das Start Event des Prozesses. Mit diesem Event beginnt der Prozess. Anschließend folgt er automatisch dem Sequenzflow zum ersten Task. Dieser Task ist ein User Task. Ein User Task benötigt die Eingabe einer Person, damit der Prozess fortgeführt wird. Nachdem der Mitarbeiter die entsprechende Eingabe vollzogen hat, splittet sich der Prozessvorgang durch eine Parallel Gateway. Zum einen wartet der Mitarbeiter auf das Resultat des Backupvorgangs und zum anderen beginnt das System mit seiner ersten Aktivität. Das Element, durch das der Mitarbeiter wartet, ist ein Intermediate Event. Intermediate Events sind Zwischen-Elemente, die auf entsprechende Ereignisse warten. Im Beispiel wäre dies eine „Nachricht“ vom System.

Die erste Aktivität des Systems wird durch ein Service Task abgebildet. Ein Service Task führt einen bestimmten Befehl aus, welcher keine Benutzereingabe benötigt. Nach Abschluss des Befehls schreitet der Prozess ohne Eingreifen eines Benutzers voran. Im Beispiel ist das nächste Element wieder ein Intermediate Event. Diesmal wartet der Prozess auf das Ablaufen einer bestimmten Zeit, hier die Backup-Zeit.

Nachdem Ablauf der Backupzeit, wird geprüft ob das Backup erfolgreich war oder ob nicht. Diese Untersuchung wird durch das Exclusive Gateway dargestellt. Je nach Resultat wird ein anderer Sequenzflow eingeleitet. Bei „nicht erfolgreich“ wird zunächst eine Fehlermeldung erstellt (Service Task) und anschließend wie bei „erfolgreich“ eine Nachricht an den Mitarbeiter produziert und geschickt (Mail Task). Nachdem dieser Mail Task erfolgreich beendet wurde, wird der Prozess wieder zusammengeführt (Parallel Gateway). Die System-Lane hat in diesem Prozess keine Aufgaben mehr, weswegen es durch das End Event beendet wird.

Die Nachricht, die an den Mitarbeiter gesendet wird, lässt das Nachrichten Intermediate Event fortfahren. Der Mitarbeiter muss anschließend die Nachricht noch bestätigen, bevor auch diese Lane beendet wird.

Mithilfe dieses Prozesses lassen sich die Elemente in verschiedene Klassen unterteilen. Zum einen gibt es die Klasse „Tasks“ oder auch „Aktivitäten“. Hier wird unterschieden zwischen Tasks die eine Benutzereingabe benötigen (User Task), um den Prozess fortzuführen, und Tasks die ohne Eingabe den Prozess automatisch fortführen (Mail und Service Task).

Des Weiteren gibt es Flows, welche den Verlauf des Prozesses symbolisieren (Sequenzflow) und Entscheidungsknoten (Exclusive Gateway), die durch bestimmte Bedingungen den weiteren Verlauf des Prozesses bestimmen.

Zwischen den einzelnen Elementen können Intermediate Events (Nachrichten oder Zeit) zwischengeschalten werden. Diese Elemente symbolisieren den Wartezustand einer Lane. Damit dieser Zustand aufgelöst wird, muss das entsprechende Ereignis eintreten.

Während der Ausführung eines Prozesses können auch mehrere Aktivitäten gleichzeitig stattfinden. Damit dies passiert, gibt es Elemente, die den Prozess splitten (Parallel Gateway). Diese Splittung muss in der Regel wieder zusammengeführt (Join) werden. Zu guter Letzt muss jede Lane noch abgeschlossen werden (End Event). Weitere nutzbare Elemente sind im Anhang C aufgeführt.

2.6 Workflow Management Coalition

Der Aufbau eines BPMS unterliegt keiner Richtlinie, allerding stellt die Workflow Management Coalition (WfMC) ein Referenzmodell zur Verfügung, welches eine Vorlage für den Aufbau bietet.

Die WfMC wurde 1993 gegründet mit dem Ziel ein Referenzmodell für BPMS zu gründen. Sie ist eine „[…] global organization of adopters, developers, practitioners, and research groups engaged in Workflow, Business Process Management (BPM) and Adaptive Case Management, as well as business process simulation and optimization.“[20]

[20] [WUU13] (WfMC – Über Uns, 10.07.2013 – 15:05)
Übersetzung: […] globale Organisation von Anwendern, Entwickler, Praktiker und Forschungsgruppen, die tätig in Workflow, BPM und Adaptive Case Management sowie in Geschäftsprozesssimulation und –optimierung sind [eigene Übersetzung]

Ein Anspruch der WfMC ist, dass eine einwandfreie Kommunikation der Schnittstellen von unterschiedlichen Herstellern gewährleistet wird. Dabei definiert sie ein BPMS in unterschiedliche Funktionsbereiche. Diese sind das Erstellen, Bearbeiten und Kontrollieren von Prozessen.[21] Abbildung 4 zeigt das Referenzmodel, mit den von der WfMC definierten Schnittstellen und deren angesprochenen Tools.

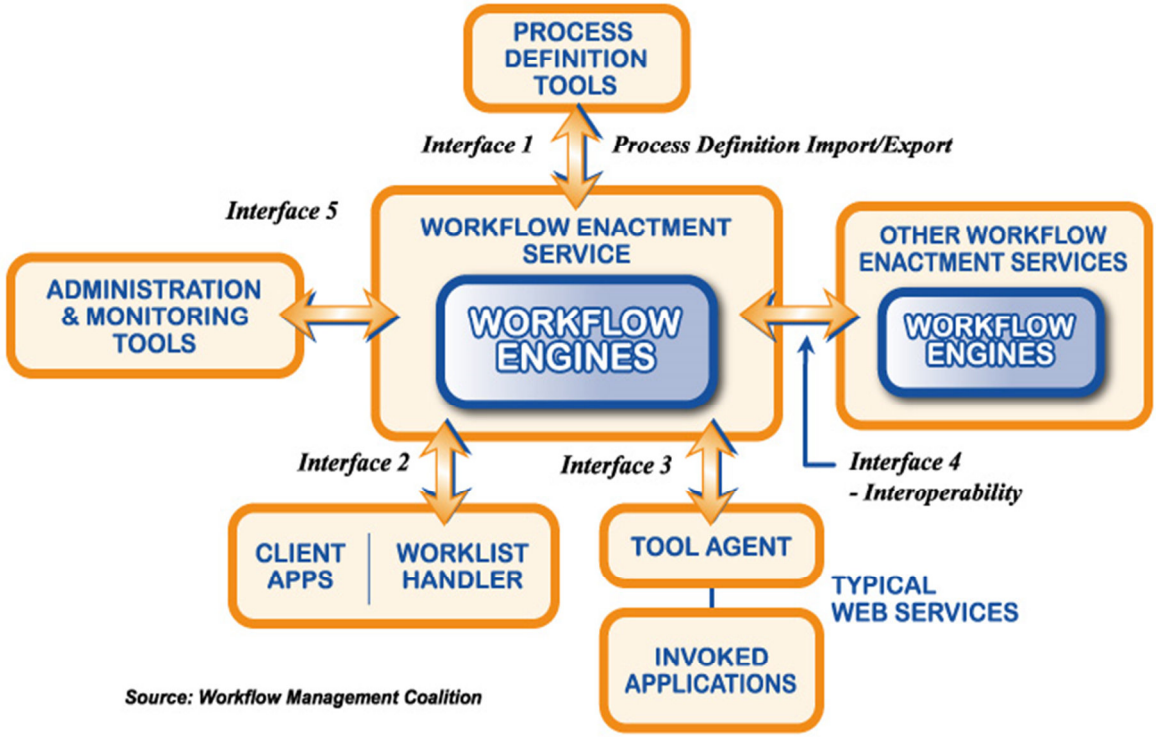


Abbildung 4: Das Workflow Referenzmodell der WfMC[22]

Allerdings muss ein BPMS nicht alle diese Tools unterstützen.[23] Falls ein Hersteller eine dieser Komponenten nicht implementiert hat, so will die WfMC, dass standardisierte Schnittstellen innerhalb der Tools zur Verfügung stehen, die es dem User erlauben, diesen Mangel durch andere Software oder Eigenentwicklung zu beheben.

2.7 Vor- und Nachteile eines Business Process Management System

Freund ([FRE10] S.16f) und Müller ([MUL05] S.24ff) geben einen Einblick worauf ein Unternehmen achten muss. Sie beschreiben Vorteile, aber auch Herausforderung, die bei der Einführung eines BPMS aufkommen.

[21] [MUL05] (Müller J. , 2005, S. 17)
[22] [WRM13] (WfMC - Refernzmodell, 17.08.2013 – 16:12)
[23] [MUL05] (Müller J. , 2005, S. 11)

Zunächst geben beide an, dass BPMS eine Brücke zwischen dem Wirtschafts- und IT-Bereich im Unternehmen bildet. In diesem Zusammenhang streben Unternehmen vor allem eine optimale Automatisierung der Geschäftsprozesse und damit eine Verkürzung der Bearbeitungszeit an. Außerdem steuern sie die Schaffung einer besseren Informationsgrundlage sowie die Steigerung der Effizienz und Wirtschaftlichkeit an. Um dies zu realisieren, muss das Unternehmen über die entsprechenden Ressourcen verfügen. Vor allem im IT-Bereich ist dies von Bedeutung, da für die Datenspeicherung große Mengen an Speicherplatz und für die Datenverarbeitung eine gute Rechenleistung benötigt wird.

Des Weiteren bietet der Einsatz des BPMS die Möglichkeit der Prozessoptimierung, wodurch Risiken frühzeitig erkennbar sind und somit das Risikomanagement verbessert wird. Die Simulationskomponente des BPMS kann dabei die nötigen Funktionen zur Verfügung stellen. Anhand der Datenanalyse findet die Prozessoptimierung, z.B. durch den Kontinuierlichen Verbesserungsprozess, statt.

Müller ([MUL05] S. 25f) beschreibt als Beispiel die Probleme innerhalb eines Geschäftsprozesses bei der Einhaltung von Fristen (z.B. termingerechte Abgabe von Dokumenten), dem Vergessen von Aufgaben oder der Nichtübermittlung von Informationen. Bei den Zielen von BPMS erwähnt er unter anderem die Verbesserung der Kommunikation, bessere Bereitstellung von Dokumenten und Formularen sowie den Einsatz von präzisem Prozessmanagement.

Bei der Optimierung von Prozessen muss sich das Unternehmen bewusst sein, dass eine Optimierung in die falsche Richtung die Gefahr einer Insolvenz, die Übernahme durch andere Unternehmen oder eine willkürliche Kundenbetreuung (vgl. [FRE10] S.17) beherbergt.

Je nach BPM-Engine (siehe Kapitel 2.8) können, anfangs hohe Kosten für den Einsatz auftreten, vor allem bei der Anschaffung neuer Software oder Hardware. Auch für die Mitarbeiter findet eine Umstellung im Unternehmen statt. Diese müssen geschult und mit dem neuen Ablauf des Prozesses vertraut gemacht werden.

Zu guter Letzt werden durch BPMS viele ISO 9000ff zertifizierte Arbeitsverfahren sichergestellt, wie z.B. dass die Ausführung eines Prozesses mit dessen Dokumentation übereinstimmt.

Zusammengefasst bietet der Einsatz von BPM im Unternehmen viele Vorteile. Allerdings muss jedes Unternehmen selbst entscheiden, in wie weit der Einsatz von BPM notwendig und erstrebenswert ist, damit die beschriebenen Risiken nicht eintreten.

2.8 Business Process Management System - Einteilungskriterien

BPMS werden nach Freund in vier Kategorien eingeteilt ([FRE10] S.15f), wonach jedes Unternehmen für sich selbst entscheiden muss, welches Produkt für das Unternehmen am geeignetsten ist.

Zunächst gibt es die **Pure Play BPMS.** Diese Lösungen integrieren die Funktionalitäten: Prozessmodellierung, Business Rule Engine, Daten- und Systemintegration, WfM, Simulation, Monitoring und Analyse in einer Lösung[24]. Sie sind so vielfältig nutzbar und umfangreich, dass ein gewisses Know-how notwendig ist und umfangreicher Support vom Hersteller angeboten wird. Die Hersteller bieten diese Lösungen meist als Black-Box an, wodurch eine Herstellerabhängigkeit entsteht.

Das zweite System sind die **Embedded BPM-Systeme**. Diese Produkte „sind in der Regel keine eigenständigen Produkte, sondern Komponenten innerhalb funktionaler Softwaresysteme, die der Workflow-Steuerung oder dem Customizing dienen“[25] Die Systeme bieten eingeschränkte Funktionalitäten, stellen aber Schnittstellen für weitere Funktionen zur Verfügung.

Die neuesten Lösungen sind die sogenannten **Software as a Service**-Systeme. Dabei stellt der Servicegeber den Servicenehmer Ressourcen und operative Dienstleistungen zur Verfügung, auf die der Servicenehmer Zugriff hat. Diese Lösungen bieten für Unternehmen meist einen kostengünstigen Einstig in BPMS. Support ist gewährleistet, allerdings kann es zu Datenschutz- und Anbindungsproblemen kommen, da die Nutzung von BPMS beim Servicegeber liegt.

Die letzte vorgestellte Lösung sind die **Open-Source-BPMS**. Diese Produkte bieten einen offenen Quellcode und weitgehende Flexibilität. Es gibt keine Herstellerabhängigkeit und die BPMS-Produkte sind zum großen Teil in Java programmiert (damit Betriebssystemunabhängig). Bei Fragen stehen dem Unternehmen meist Foren oder offener Quellcode zur Verfügung. Diese Lösungen haben keinen, begrenzten oder kostenpflichtigen Support.

[24] [SCH00] (Schnägelberger & Besch, 17.07.2013 - 20:35)
[25] [FRE10] (Freund, 2010, S. 15)

3. Activiti

Auch wenn die BPMS sich zum Großteil an das Referenzmodell halten, sind sie untereinander verschieden strukturiert und auf andere Funktionen spezialisiert. Dies verhindert eine Verallgemeinerung bezüglich der Integration. Activiti steht trotzdem als Repräsentant für andere BPMS, um zu zeigen, wie eine Integration funktioniert und was dabei zu beachten gilt.

3.1 Die Entwicklung von Activiti

Activiti ist ein open source BPM-System, dessen Entwicklung 2010 begann und von dem im selben Jahr ein erstes Alpha-Release auf den Markt gebracht wurde. Die Entwicklung von Activiti begann direkt nach dem Wechsel der beiden Hauptentwickler von jBPM, Tom Baeyens und Joram Barrez, zu Alfresco. Zusammen mit den Entwicklern von Alfresco wurde Activiti quelltextunabhängig von jBPM entwickelt.[26]

Die erste Version von Activiti wurde Activiti 5.0 genannt, da auf die Vorentwicklungen von jBPM 1.0 – 4.0, auf dessen Basis Activiti entstand, aufmerksam gemacht wurde. Die aktuelle Version von Activiti ist Activiti 5.13 (Stand 27.07.2013). Auf diese Version baut das Kapitel 6, mit all seinen Konfigurationen und den Quelltextbeispielen, auf. Firmen, wie Alfresco, springsource, Next Level Integration, Salves[27] waren oder sind bei der Weiterentwicklung von Activiti involviert.

3.2 Aufbau und Komponenten

Activiti besteht aus einem Java Source Code, welcher ab JDK (Java Development Kit) 6 ausführbar ist.[28] Es bietet sechs Komponenten an, die aus Activiti eine Komplettlösung für das BPM im IT-Bereich bilden.[29] Activiti unterstützt sechs Datenbankmanagementsysteme (siehe Anhang G[1-e]) und führt die Transaktionen über den Object Relational Mapper (ORM - siehe Kapitel 4.3.2) iBatis aus.

> *„Eine Transaktion ist die Bearbeitung von einer oder mehreren miteinander verknüpften Funktionen, die anhand einer logischen Verarbeitungsfolge zusammengefasst sind.“*[30]

[26] [RAD12] (Rademakers, 2012, S. xvi)
[27] [ACT13] (Activiti, 29.07.2013 – 8:43)
[28] [AUG13] (Activiti User Guide, 29.07.2013 – 15:24)
[29] [ACC13] (Activiti, 17.08.2013 – 15:36)
[30] [MUL05] (Müller J. , 2005, S. 56)

Transaktionen haben vier entscheidende Eigenschaften (vgl. [MUL05] S 56). Sie sind atomar, konsistent, isoliert und beständig. Das heißt:

> *„Eine Transaktion bezeichnet eine zusammenhängende Folge von Bearbeitungsschritten, die entweder komplett oder gar nicht ausgeführt werden.“*[31]

Activiti unterteilt seine Komponenten in drei Tool-Gruppen (siehe Abbildung 5):

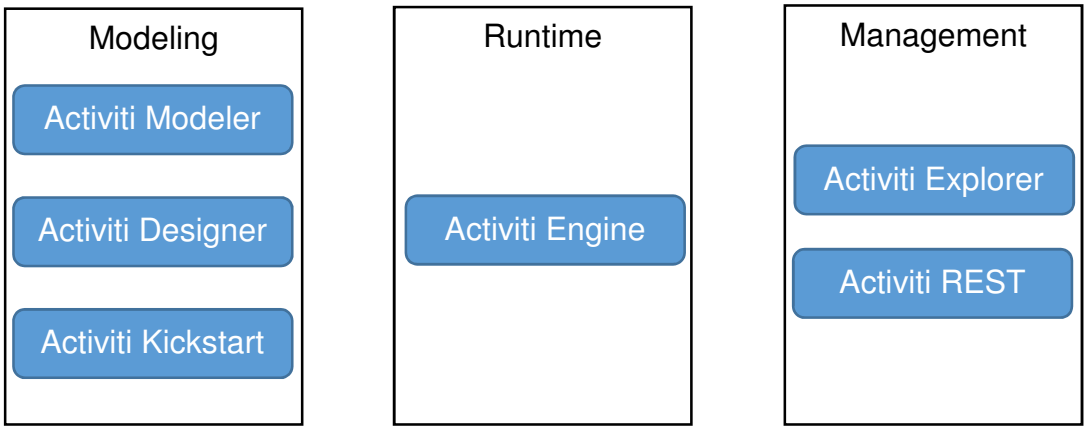


Abbildung 5: Aufbau und Komponenten von Activiti[32]

3.2.1 Runtime

Von den drei Tool-Gruppen beinhaltet Runtime die wichtigste Komponente, die Activiti Engine. Sie umfasst Funktionen, wie das Lesen und Ausführen von BPMN 2.0 Geschäftsprozessen oder die Erstellung von Aufgaben (Tasks).[33] Die Engine unterstützt dazu eine Fülle von Java Laufzeitumgebungen, wie Spring oder Java Transaction API (JTA), arbeitet Standalone oder Embedded, beinhaltet spezielle Listener und Handler.

Die Installation der Engine als Standalone oder Embedded kann bei [AUG13] nachgelesen werden. Die Activiti-Engine kann über den Spring-Applikation-Kontext gestartet werden, wodurch es möglich ist die Dependency Injection (siehe Kapitel 4.3.1) auszuführen oder Spring Beans im BPMN-Prozess zu nutzen. Die Verwaltung dieses Tools übernimmt der Prozess-Engineer.

[31] [MUL05] (Müller J. , 2005, S. 231)
[32] [ACC13] (Activiti, 29.07.2013 – 09:16)
[33] [RAD12] (Rademakers, 2012, S. 5)

3.2.2 Modeling

Die zweite Tool-Gruppe ist das Modeling. Diese weist alle Komponenten auf, welche es ermöglichen, eine Activiti relevante Prozessdefinition zu erstellen. Das Anfertigen übernimmt der Prozess-Analyst oder der Prozess-Engineer. Er hat drei spezifische Komponenten zur Auswahl.

Activiti Designer

Eine dieser Komponenten ist der Activiti Designer. Zahn ([ZAH13]) gibt einen ausreichenden Einblick in die Funktionen und den Aufbau des Activiti Designer. Die Installation und detaillierte Benutzungsmöglichkeiten bietet das Buch ([RAD12]) und der UserGuide auf der Activiti Homepage ([AUG13]) Hier sei nur nochmal erwähnt, dass es sich um ein Eclipse-Plug-In handelt, welches über eine grafische Oberfläche die Erstellung einer Prozessdefinition ermöglicht.

Activiti Modeler

Eine weitere Komponente ist die webbasierte Modellierungsumgebung Activiti Modeler. Es ist eine open source Lösung der Firma „keep it simple“-BPM und ähnelt stark dem Signavio Process Editor (siehe Anhang E). Der Activiti Modeler beherbergt dieselben Funktionen wie der Activiti Designer.

Activiti Kickstart

Die letzte Komponente, die das Modeling enthält, ist der Activiti Kickstart. Dieser ist ebenfalls eine webbasierte Modellierungsumgebung, ist aber in den aktuelleren Versionen von Activiti nicht mehr enthalten, sondern nur bis Version 5.6 verfügbar.

3.2.3 Management

Die letzte Tool-Gruppe ist das Management. In ihr sind die Komponenten Activiti REST und Activiti Explorer enthalten. Diese Komponenten dienen der Verwaltung von Prozessen. Darunter zählen das Monitoring, die Simulation sowie das Reporting. Außerdem sind die Prozess-Beteiligten, der Prozess-Administrator sowie der Prozess-Engineer und –Manager.

Activiti REST

Activiti REST (Representational State Transfer) ist eine Web-Applikation, die eine REST-Schnittstelle zur Activiti Engine bietet. REST bezeichnet „[…] eine Architektur für Webanwendungen, in der eine URL genau ein Seiteninhalt als Ergebnis einer serverseitigen Aktion darstellt.“[34] Sie ist bei der Standardinstallation der Einstiegspunkt zur Activiti Engine und ermöglicht die Bearbeitung der Engine-Konfiguration.[35]

Activiti Explorer

Der Activiti Explorer ist eine Web-Applikation, die ein breites Spektrum an Funktionen in Verbindung mit der Activiti Engine liefert.[36] Es bietet unter anderem die Möglichkeit, die durch das Modeling erstellten Prozessdefinitionen zu starten, diese Prozesse zu testen und zu überwachen (Monitoring). Gleichzeitig steht auch eine Simulation der Prozesse zur Verfügung.

3.2.4 Beziehung zum Referenzmodell der Workflow Management Coalition

All diese Komponenten benötigen eine Möglichkeit, um auf die Engine von Activiti zuzugreifen und ihre Operationen auszuführen. Die folgende Tabelle zeigt, welche Activiti-Komponenten standardmäßig über welche Interfaces aus dem WfMC-Referenzmodell auf die Activiti Engine zugreifen.

Tool	Interface	Komponente
Process Definition Tools	Interface 1	Activiti Modeler
Workflow Client Application	Interface 2	Activiti Explorer / Activiti REST
Invoked Application	Interface 3	Activiti Explorer / Activiti REST
Workflow Interoperability	Interface 4	Nicht Dokumentiert
Administration & Monitoring Tools	Interface 5	Activiti Explorer

Tabelle 1: Activiti und WfMC-Referenzmodell

3.3 Integrationsvarianten

In Kapitel 2.8 wurden die BPMS in verschiedene Typen untergliedert. Activiti ist sowohl ein Open-Source, als auch ein Embedded BPMS. Um Activiti zu nutzten, muss der Entwickler sich zunächst entscheiden, wie Activiti eingesetzt wird. Activiti unterstützt sowohl die Embedded Implementierung, als auch die Standalone Implementierung.

[34] [WIR13] (Wikipedia-Rest, 29.07.2013 – 14:53)
[35] [RAD12] (Rademakers, 2012, S. 5)
[36] [RAD12] (Rademakers, 2012, S. 5)

Des Weiteren ist es wichtig zu wissen, dass Activiti als Embedded oder Standalone Service in den JBoss Application Server installierbar ist. Dadurch wird die Anwendung in einer Java Platform, Enterprise Edition (JEE)–Applikationen möglich.[37]

Embedded

Embedded wird genutzt, um in der eigenen Applikation eine Instanz der Activiti-Engine zu bilden. Der Zugriff auf die Methoden und Funktionen der Engine erfolgt über die Activiti Java API. Dabei kann es sich um eine Web-Applikation, einen Anwendungsserver oder eine Java Client Applikation handeln. Die Anwendung der Engine ist so gedacht, dass sie in derselben Java Virtual Machine läuft, wie die Applikation.[38] Abbildung 6 zeigt den Aufbau der Embedded-Variante.

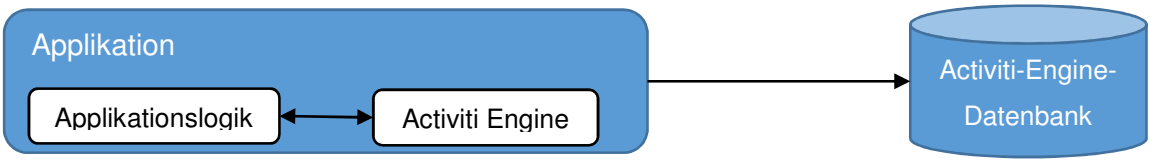


Abbildung 6: Deploymentdiagramm der Activiti-Engine Embedded[39]

Standalone

Bei der Standalone-Einrichtung wird, durch die REST-API, mehreren Anwendungen der Zugriff auf die Activiti-Engine ermöglicht. Gestartet wird die Engine über einen Webserver, wie Apache Tomcat, im Activiti Explorer oder der REST-Webapplikation.[40] Abbildung 7 zeigt den strukturellen Aufbau zwischen den Applikationen und der Engine.

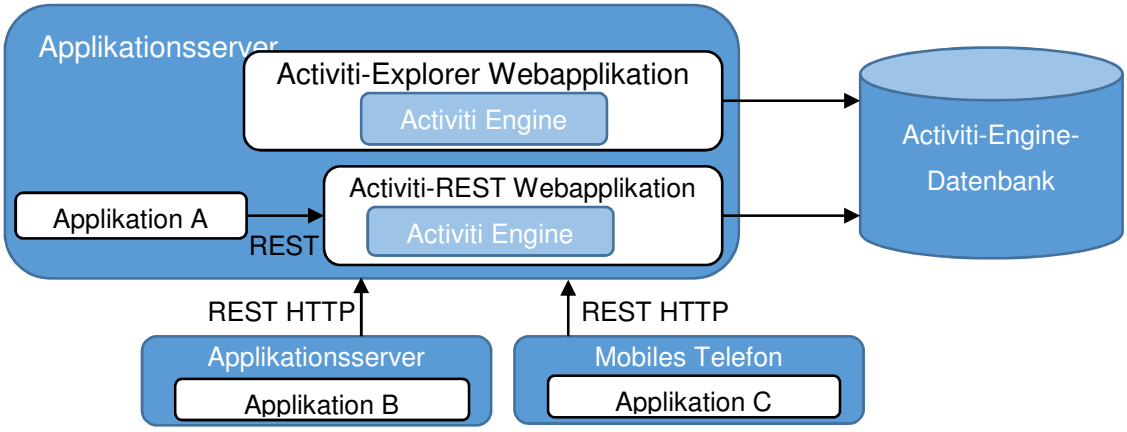


Abbildung 7: Deploymentdiagramm Activiti-Engine Standalone[41]

[37] [RAD12] (Rademakers, 2012, S. 221)
[38] [RAD12] (Rademakers, 2012, S. 170)
[39] [RAD12] (Rademakers, 2012, S. 171)
[40] [RAD12] (Rademakers, 2012, S. 170)
[41] [RAD12] (Rademakers, 2012, S. 173)

3.4 Zusammenfassung

Die Analyse zeigt, dass Activiti ein in Java programmiertes BPM-System ist, welches als Embedded oder Standalone Applikation benutzt werden kann. Es bietet eigene Komponenten, zur Erfüllung der BPMS-Funktionen.

Je nach benötigter Geschäftsanwendung, können entweder die Komponenten von Activiti benutzt werden oder eigene Komponenten, welche über die entsprechenden Schnittstellen auf die Activiti Engine zugreifen. Activiti besitzt seine eigenen Datenbanktabellen und führt die Transaktionen über iBatis aus. Zu guter Letzt unterstützt es Spring.

4. Geschäftsanwendung

> *Geschäftsanwendung, Business Software oder Business Application ist jede Art von Software oder Computertools die benutzt werden, um verschiedene Geschäftsfunktionen auszuüben.*[42]

Jede Geschäftsanwendung beherbergt eine gewisse Komplexität, die von Entwickler für die Entwickler und die Kunden beherrschbar bleiben muss. Anwendungen die nicht kontrollierbar sind, können nicht verändert werden, ohne weitere Probleme innerhalb der Anwendung zu erzeugen.

Die Komplexität wird in der Wirtschaft durch die Menge der Details repräsentiert. Je höher die Detaillierung, ohne dass eine Vereinfachung möglich ist, desto komplexer ist der Sachverhalt.[43] Der Sachverhalt ist ein System und kann aus mehreren Systemelementen mit unterschiedlichen Detaillierungsgrad bestehen. Der Detaillierungsgrad einer Software bestimmt sich aus der Anzahl der Systemelemente und deren Beziehungen untereinander (Kopplungen). Weitere Faktoren des Detaillierungsgrad sind das Maß an Wissen für die kausalen Zusammenhänge sowie die Reduktionsmöglichkeit der Komplexität von Systemelementen.[44]

Außerdem bestimmt sich die Komplexität unter anderem durch die Mehrschichten-Architektur, Datenhaltung, Graphical User Interface, Spring oder Abhängigkeiten zwischen den Komponenten.

4.1 Mehrschichten – Architektur

Die Architektur einer Anwendung trägt zur Komplexität bei, denn je nachdem wie und wo in einer Anwendung Logik, Datenhaltung und Darstellung implementiert werden, kann zwischen einem Zwei-, Zweieinhalb-, Drei- und Vier-Schichten-Modell unterschieden werden.[45] Dies sind die bekanntesten und am meisten genutzten Modelle. Da Leasman ein Drei-Schichten-Modell ist (siehe Abbildung 11), wird sich auf diese Art der Softwarearchitektur beschränkt.

[42] [WBS13] (Wikipedia -Business Software, 23.08.2013 – 15:16)
[43] [WIK13] (Wikipedia - Komplexität, 26.08.2013 – 08:45)
[44] [FEE13] (Feess, 26.08.2013 – 09:18)
[45] [HAA06] (Haag, 2006, S. 8 ff)

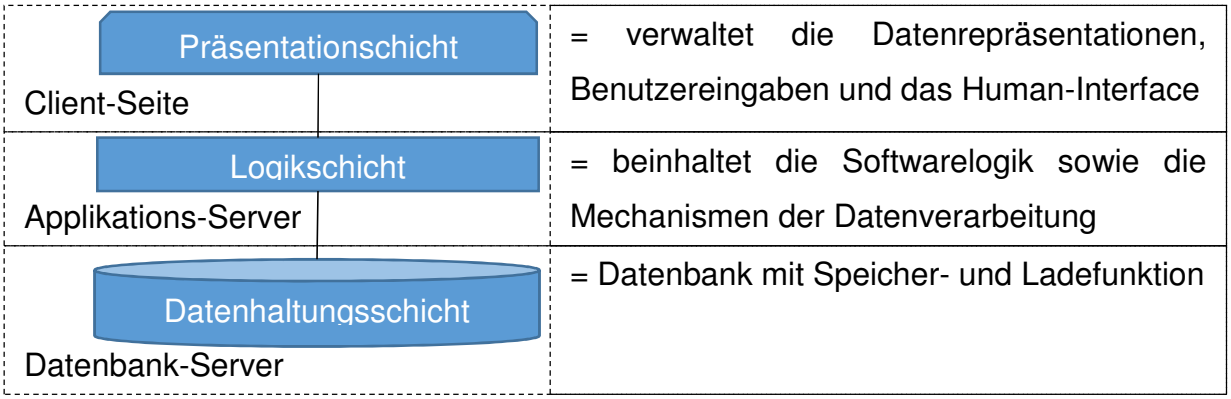


Abbildung 8: Drei-Schichten-Modell

Das Drei-Schichten-Modell (siehe Abbildung 8) besteht allgemein aus 3 Ebenen:

Die Vorteile dieser Art der Architektur liegen vor allem darin, dass der Applikation Server die Datenbankzugriffe verwaltet und eine Trennung der Geschäftslogik mit der Präsentationslogik stattfindet. Dadurch kann jede Schicht für sich betrachtet und verwaltet werden, was wiederum die Komplexität beherrschbar macht und die Geschwindigkeit der Anwendung erhöht. Außerdem reduzieren sich durch die Geschäftslogikschicht die Abhängigkeiten innerhalb des Systems.

Allerdings ist eine Clientinstallation nötig und die Geschäftslogik muss Anfragen mehrerer Clients bearbeiten, was die Komplexität erhöht.

4.2 Java Platform Enterprise Edition

Für die Umsetzung eines Drei-Schichten-Modells wurde von Sun Microsystems die Java Platform Enterprise Edition (JEE) entwickelt. Zwei Definitionen beschreiben, welche Bedeutung JEE hat.

> *„Java EE Platform is a standard platform for hosting Java EE applications“*[46]

und

> *„JEE […] ist eine Spezifikation für eine Softwarearchitektur zur transaktionsbasierten Ausführung von in Java programmierten Anwendungen.“*[47]

[46] [CHI09] (Chinnici & Shannon, 2009, S. 2)
Übersetzung: JEE ist eine Standard-Platform zur Bereitstellung von JEE-Applikationen [eigene Übersetzung]
[47] [WIJ13] (Wikipedia - JEE, 14.08.2013 – 12:32)

JEE stellt also eine Standardarchitektur bei der Entwicklung von Geschäftsanwendungen, mit dem Ziel, die Kosten und Komplexität bei der Erstellung von Geschäftsanwendungen zu senken.[48] Sie beinhaltet standardisierte APIs, ist aber selbst weder ein Produkt, noch ein System.

Abbildung 9 zeigt den Aufbau der JEE Version 6 Container und deren Beziehungen untereinander. Dabei stellen die einzelnen Container selbstständige Laufzeitumgebungen dar, die entsprechende Dienstleistungen (untere Hälfte eine Container) zu erbringen haben. Des Weiteren decken diese Container betriebssystemgebundene Probleme, wie Netzwerknutzung, ab. Dadurch kann sich der Entwickler voll auf die Geschäftslogik konzentrieren, was die Komplexität senkt. Die Datenhaltung erfolgt in einer Datenbank, welche über die Java Database Connectivity-API zugänglich ist.[49]

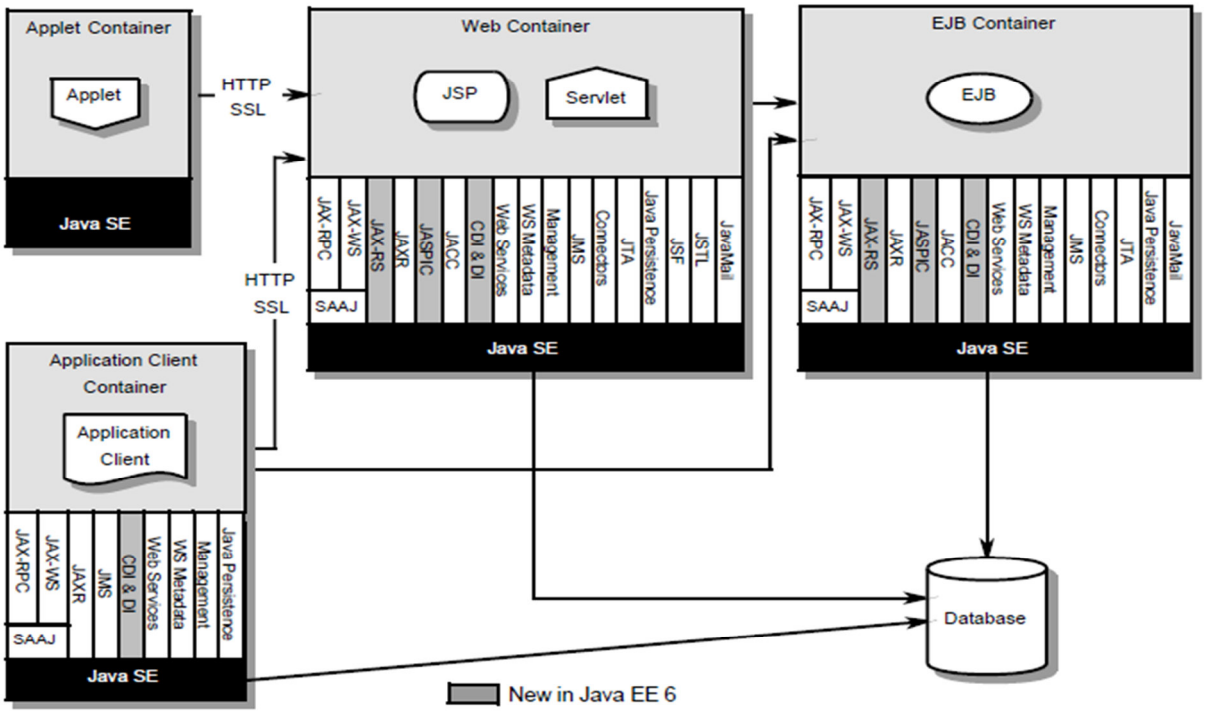


Abbildung 9: Architekturdiagramm JEE 6[50]

Die JEE hat klar definierte Schnittstellen, zwischen den Containern und Komponenten, die eine Interoperabilität verschiedener Softwarekomponenten ermöglicht.[51]

[48] [CHI09] (Chinnici & Shannon, 2009, S. 1)
[49] [CHI09] (Chinnici & Shannon, 2009, S. 11)
[50] [CHI09] (Chinnici & Shannon, 2009, S. 6)
[51] [CHI09] (Chinnici & Shannon, 2009, S. 5)

4.3 Leasman

Der Leasman ist eine komplexe Geschäftsanwendung, da jeder Service (im Leasman die Komponenten, siehe Kapitel 4.3.1) als ein Systemelement betrachtet werden kann, welcher aufgrund der Beziehungen zu den anderen Service, einen sehr hohen Detaillierungsgrad aufweist. Der Leasman beinhaltet mehr als 50 solcher Service, mit unzähligen Parametern. Die daraus resultierte Menge an verschiedenen Benutzereingaben und Kombinationen für Kopplungen zwischen den Service (momentan mehr als 150), legen die Komplexität der Software nahe.

Auch nutzt der Leasman die JEE Version 6 und implementiert an den JEE-Schnittstellen seine eigenen Funktionen. Die Container werden dabei wie folgt genutzt:

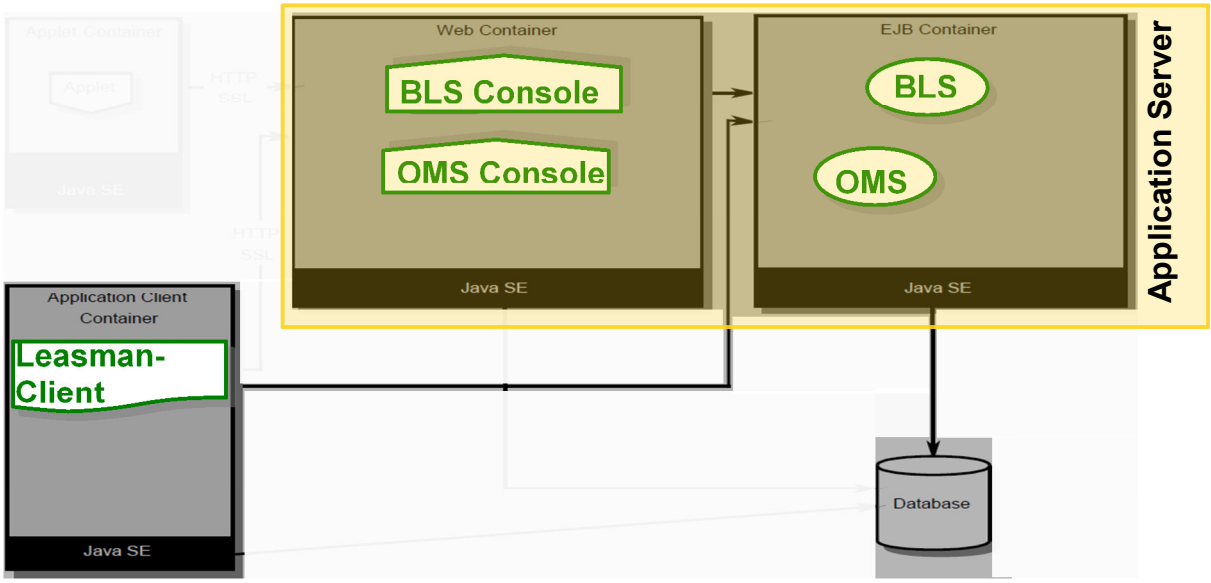


Abbildung 10: JEE 6 im Leasman[52]

Der in Abbildung 10 aufgeführte Application Server kann im Leasman durch jeden Applikationsserver, der die Voraussetzungen des JEE erfüllt, abgebildet werden. Standardmäßig wird der JBoss Application Server 7 genutzt.

[52] * OMS = Output Management System BLS = Business Logic Server

Da der Leasman als Grundlage JEE verwendet, wird ebenfalls ein Drei-Schichten-Modell realisiert (siehe Abbildung 11).

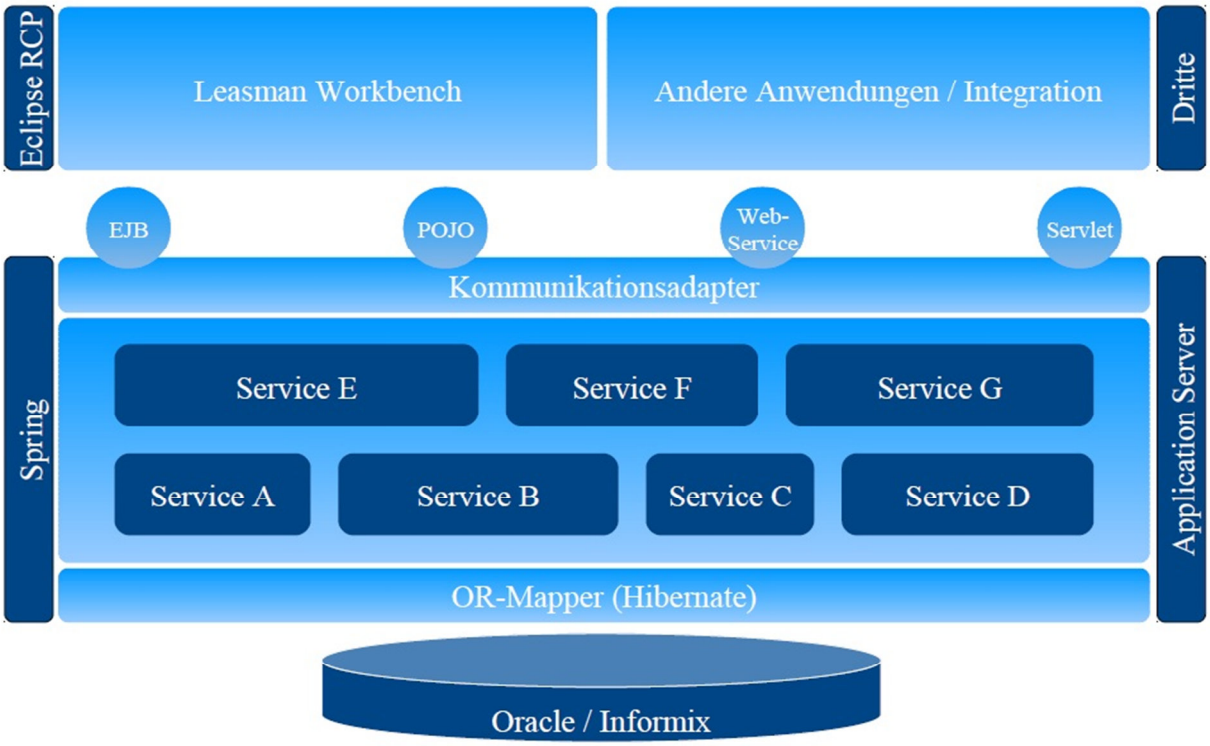


Abbildung 11: Leasmanarchitektur[53]

An die Client-Seite des Leasman kann die Leasman-Workbench oder eine beliebige andere Anwendungen angeschlossen werden. Das Eclipse Rich Client Plattform (RCP)-Plug-In unterstützt die Entwicklung der Benutzeroberfläche. Das RCP „stellt ein Grundgerüst bereit, das um eigene Anwendungsfunktionalität erweitert werden kann.“[54]

Um den Leasman in Module aufzuteilen und diese untereinander abzugrenzen wird das OSGi-Framework (Open Service Gateway initative) genutzt. Das OSGi erleichtert die Modularisierung und Verwaltung von Diensten und Anwendungen per Komponenten(Service)-Model.[55] Es besteht aus Spezifikationen, welche dynamische Komponenten für Java definieren. Die Spezifikationen ermöglichen ein Entwicklungsmodell, bei denen Anwendungen aus verschiedenen Komponenten zusammengesetzt sind. Die Komponenten können ihre Implementierung gegenüber anderen Komponenten verbergen, kommunizieren aber über spezielle Dienste.[56]

[53] [DEA13] (DELTA Proveris AG , 2013)
[54] [EBE11] (Ebert, 2011, S. 3)
[55] [OSI13] (OSGi Alliance - About, 15.08.2013 – 08:59)
[56] [OSG13] (OSGi - What Is OSGi, 26.08.2013 – 15:11)

Dadurch werden die Komponenten einzeln angesprochen und verändert, ohne dass andere Komponenten die Veränderungen mitbekommen. Einzelne Module oder Plug-Ins können dementsprechend ohne größeren Aufwand, in das aktuelle Produkt, implementiert werden.

Alle Funktionen, die der Kunde ausführen kann, werden in der komponenten-orientierten Architektur (Applikationsschicht) des Leasman verwaltet. Der Application Server (JBoss) führt die Anwendungsprogramme, die der Client benötigt, aus. Dabei stellt er die Dienste für die Transaktionen, Authentifizierung sowie die Datenbankzugriffe zu Verfügung.

4.3.1 Komponentenstruktur

Die Komponenten unterliegen strikten Regeln für deren Architektur und Entwicklung. Daniel Haag beschreibt die Grundlagen für die Leasman-Mittelschicht in seiner Masterarbeit [HAA06]. Jede Komponente besitzt dabei zwei Schnittstellen. Einmal zum Ansprechen von internen Komponenten (IComponentManager) und zum anderen damit der Client Zugriff auf die Komponente hat (IComponentService). Abbildung 12 veranschaulicht die Architektur.

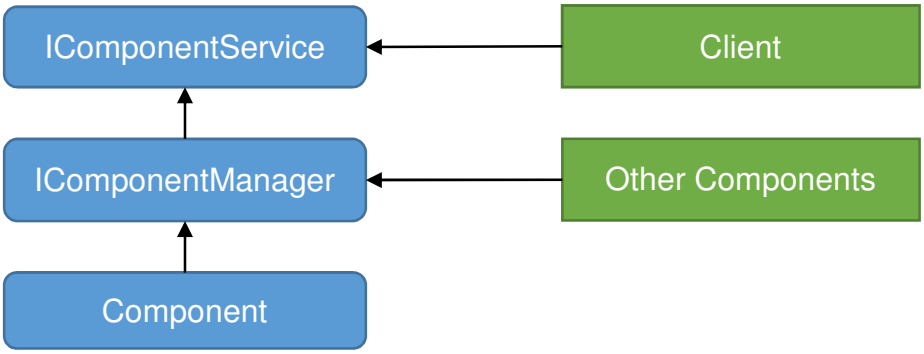


Abbildung 12: Komponentenaufbau und -zugriff

Wichtig ist dabei, dass keine Ringabhängigkeit zwischen den Komponenten auftritt. Es entsteht eine Ringabhängigkeit, wenn Komponenten untereinander, aufeinander zugreifen oder falls sie über andere Komponenten auf Komponenten zugreifen, die wiederum auf sie zugreifen (siehe Abbildung 13).

Abbildung 13: Beziehung von Komponenten ohne und mit Ringabhängigkeit

Die Programmierung ohne Ringabhängigkeiten hat den Nachteil, dass die Erstellung der Softwarearchitektur mit einem hohen Zeitaufwand verbunden ist. Allerdings kann ohne Ringabhängigkeit und durch die Kapselung der Komponenten, deren Funktionalität in verschiedenen Prozessen eingesetzt und von anderen Anwendungen genutzt werden. Des Weiteren ist die Pflege der Komponenten ohne die Ringabhängigkeit einfacher, was die Komplexität vereinfacht.

Die Komponenten des Leasman nutzen das open source Framework Spring, um ihre Einstellungen in einer XML-Datei zu definieren. Spring bietet dazu eine externe Konfiguration (z.B. Ressourcenzuweisung) von Klassen über Spring Beans. Spring Beans sind definierte Java-Objekte. Ihnen muss eine Klasse und ein Name übergeben werden. Spring nutzt die Dependency Injection, welche die Möglichkeit bietet Referenzen und Abhängigkeiten zwischen den Beans zu definieren. Die Spring-Konfigurations-XML-Datei wird beim Hochfahren des BLS gelesen und die Beans erstellt. „Für jedes Objekt, sei es Geschäftsobjekt, Mapper oder Managerklasse, wird eine Spring Bean definiert. Dies ermöglicht die Implementierung von Objekten […]“[57] Weitere Informationen können bei [WOL10] nachgelesen werden.

4.3.2 Datenbankzugriff

Innerhalb des JEE ist für den Standarddatenbankzugriff die Schnittstelle Java Persistence API (JPA)-2.0 Spezifikation definiert. Sie ist nur ein Standard-API zur Datenbankanbindung, was bedeutet, dass sie von JPA Providern, wie EclipseLink, Hibernate, iBatis oder Apache OpenJPA, implementiert wird. Dies ermöglicht eine bessere Austauschbarkeit der Provider.

[57] [HAA06] (Haag, 2006, S. 86)

Doch auch wenn der Austausch dieser Provider möglich ist, muss der Softwareentwickler bei der Nutzung der einzelnen Provider vorher überlegen, welcher für die programmierte Geschäftsanwendung am besten geeignet ist. Zum Beispiel benötigen einige Provider weiterhin SQL-Anweisungen, andere sind Datenbankabhängig oder die automatische Tabellengenerierung ist nicht immer vorhanden.

In Abbildung 11 wird die JPA Schnittstelle durch einen ORM (JPA Provider) aufgeführt. ORM erlauben eine beiderseitige Umwandlung von Java-Objekte und Relationalen-Objekte, so dass diese Objekte anschließend in der Datenbank eingefügt (Insert), gelöscht (Delete), bearbeitet (Update) sowie aus der Datenbank geladen (Load) werden.

Die DELTA proveris AG arbeitet nach dem Schema „Repository“-Pattern (siehe Anhang F) zur Ausführung dieser Datenbankoperationen und setzt das ORM „Hibernate“ auf. Es gibt weitere Schemen (vgl. [FOW03] S.33ff), die nicht näher unter Betrachtung stehen. Trotz der Nutzung des Repository muss an manchen Stellen immer wieder mit SQL-Befehlen gearbeitet werden, da sonst die Komplexität der Methoden steigt und deren Wartbarkeit sinkt. Abbildung 14 zeigt das Hibernate-Datenmodell im Leasman.

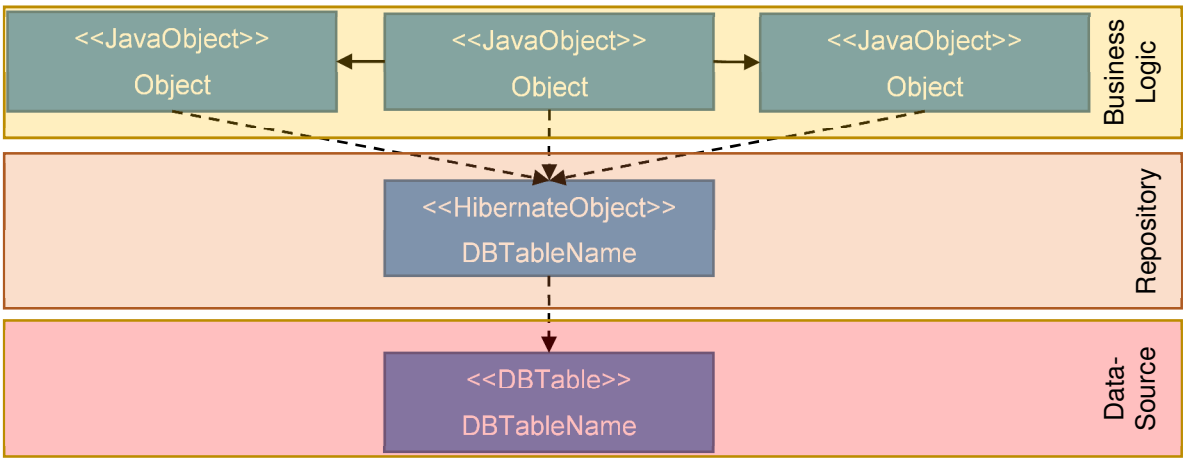


Abbildung 14: Datenmodell Hibernate mit den Geschäftsobjekten

Zusammenfassend betrachtet, wird zur Ausführung eines ORM mindestens ein Java-Objekt (Business-Objekt) benötigt, welches eine Tabelle der Datenbank abbildet. Anschließend kommt ein Hibernate-Objekt zum Einsatz, welches einen Datensatz aus der Tabelle repräsentiert. In der Datenbanktabelle wird über Hibernate das Hibernate-Objekt durch Abfragen eingefügt (Insert), gelöscht (Delete) oder bearbeitet (Update). Damit Hibernate auf die Datenbank zugreifen kann, benötigt die Geschäftsanwendung noch eine Hibernate Konfigurationsdatei.

Eine detailliertere Einführung und Beispiele, in dem der Aufbau der benötigten Dateien aufgeführt ist, kann bei [HAA06], [BAU07] und [HOR13] nachgelesen werden.

4.3.3 Datenbankaufbau

Die Datenhaltung im Leasman erfolgt in verschiedenen Datenbanken. Es gibt eine Kern-Datenbank mit Tabellen, die jeder Mandant benötigt und eine Datenbank pro Mandant. Activiti wird in der Kerndatenbank integriert, da sie jedem Mandanten zur Verfügung stehen muss (siehe Abbildung 15).

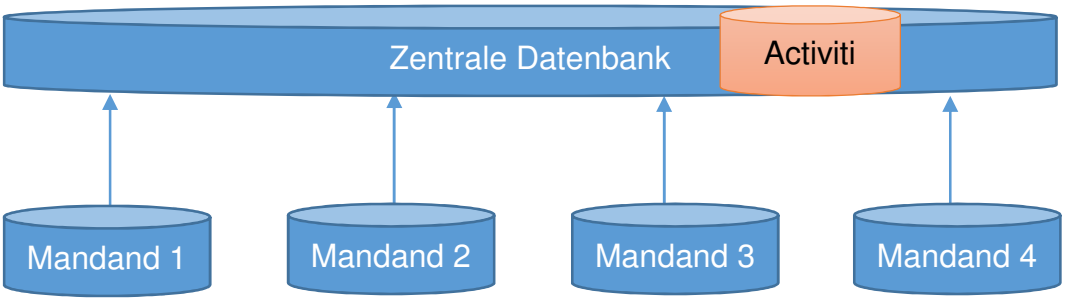


Abbildung 15: Datenbankschema des Leasman

4.4 Zusammenfassung

Der Leasman ist eine in Java programmierte Geschäftsanwendung auf Basis der JEE. Es besteht aus einem Drei-Schichten-Modell und besitzt seine eigene Präsentationschicht sowie eine Schnittstelle für kundeneigene Oberflächen. Die Komponenten der Logikschicht beinhalten die Funktionen, die ein Anwender starten kann und benutzen das Framework Spring, um die Kopplung untereinander lose zu halten. Der Aufbau einer solchen Komponente unterliegt dabei strikten Regeln. Die Datenhaltung erfolgt über Hibernate in einer Oracle- oder Informix Datenbank.

5. Integrationskonzept

In den vorangegangenen Kapiteln wurden die Grundlagen von BPM, Activiti und Geschäftsanwendungen detailliert erläutert. In diesem Kapitel erfolgt zunächst die Erörterung von Anforderungen an die Integration von Activiti in das Leasman-System. Auf Basis dieser Anforderungen wird anschließend ein Integrationskonzept konzipiert.

5.1 Anforderungen an die Integration

Die Anforderungen an das zu erstellende Integrationskonzept leiten sich aus verschiedenen Quellen ab. Zum einen resultieren Anforderungen aus den Vorgaben der DELTA proveris AG bezüglich des Produktentwicklungsplans und zum anderen aus der vorgegebenen Leasman-Systemarchitektur. Die oberste Anforderung ist die Integration von Activiti als Leasman-Komponente in die vorherrschende Systemlandschaft (siehe Abbildung 16).

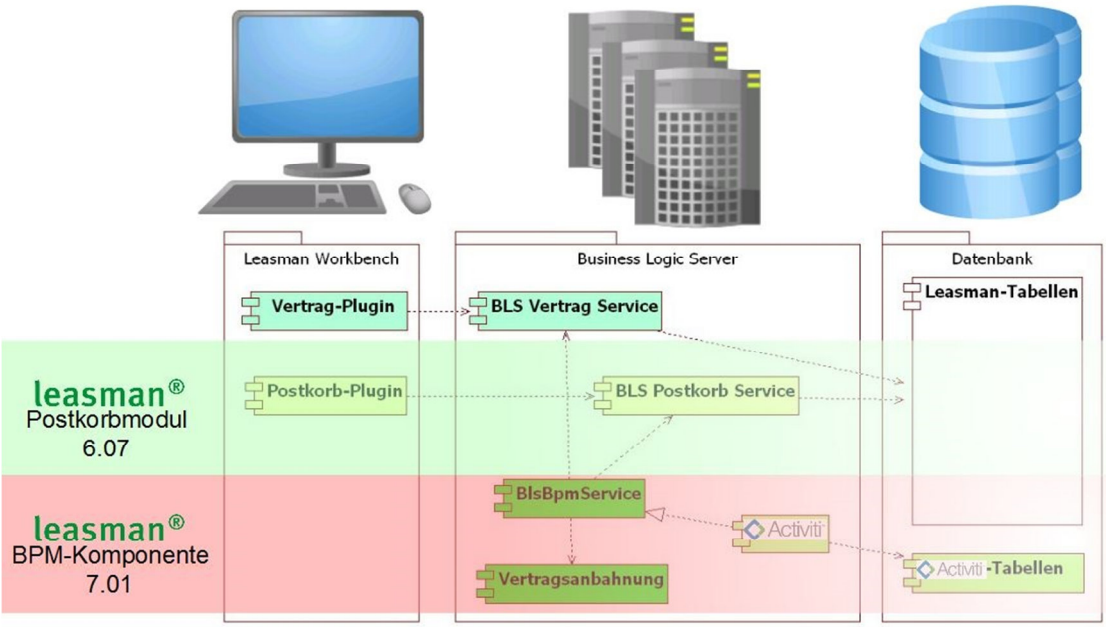


Abbildung 16: Integrationsaufbau von Activiti im Leasman[58]

Anhand dieser formulierten Anforderung lassen sich die weitere Anforderungen ableiten. Diese gliedern sich in drei Bereiche: Business Logic Server, Datenbank, Leasman Workbench.

[58] [DEB12] (DELTA proveris AG , 2012)

5.1.1 Business Logic Server

Im Kontext des BLS erheben sich die Anforderungen aus den Betrachtungen, wie eine Realisierung von Activiti als Leasman-Komponente ermöglicht wird.

A1: Engine-Integration

Wie bereits in dem vorangegangenen Abschnitt erläutert, soll Activiti als Leasman-Komponente agieren und sich in das System nahtlos integrieren.

A2: Konformität mit Leasman-BpmService

Es existiert eine Leasman-Komponente, welche von den eingesetzten BPM-Engines abstrahiert. Activiti muss diese vorgegebene Schnittstelle und die damit verbundenen Funktionalitäten realisieren.

A3: Behandlung von User Tasks

Die Activiti-Engine geht beim Erreichen eines User Task in den Wartezustand über. Im Leasman-System existiert für die Verwaltung von Aufgaben eine spezielle Komponente, der Postkorb. Beim Erreichen eines User Task muss die Activiti-Engine eine Aufgabe im Postkorb anlegen und einem Benutzer zuweisen.

A4: Transaktionsmanagement

Activiti muss das im Leasman-System bereitgestellte Transaktionsmanagement nutzen.

5.1.2 Datenbank

Aus den Betrachtungen bezüglich der zur Verfügung stehenden Datenbankumgebung leitet sich Anforderung A5 ab.

A5: Datenhaltung

Die Tabellen von Activiti müssen in die Zentraldatenbank des Leasman integriert werden. Dies begünstigt auch die Umsetzung der Forderung eines einheitlichen Transaktionsmanagements (siehe Anforderung A4).

5.1.3 Leasman Workbench

A6: Human Workflow Interface

Es wird ein Human Workflow Interface (HWI) benötigt, um die Benutzereingaben zu managen. Diese umfasst die Entgegennahme und Weiterleitung von Benutzereingaben an den User Task sowie dessen Reaktivierung nach erfolgreicher Eingabe.

5.2 Eignung der Business Process Management Engine

Anforderung	Bewertung
A1: Engine Integration	+
A2: Konformität mit Leasman-BpmService	+
A3: Behandlung von User Tasks	+
A4: Transaktionsmanagement	o
A5: Datenhaltung	o
A6: Human Workflow Interface	+

Tabelle 2: Bewertung der Anforderungen[59]

Anforderungen A4 und A5 werden von Activiti erfüllt, ohne das Anpassungen notwendig sind. Die anderen Anforderungen sind nach der Analyse des Funktionsumfangs von Activiti (siehe Kapitel 3) realisierbar, jedoch sind dafür verschiedene Anpassungen notwendig.

In den folgenden Abschnitten erfolgt die Konzipierung einer Lösung, um die offenen Anforderungen umzusetzen.

5.3 Vorstellung des Integrationskonzeptes

Die Realisierung der Anforderung A1 erfordert die Erstellung eines Adapters, welcher den Betrieb von Activiti als Leasman-Systemkomponente ermöglicht. Dieser implementiert die geforderten Schnittstellen jeder Leasman-Komponente (siehe Kapitel 4.3.1). Im Kontext dieses Adapters erfolgt ebenfalls die Umsetzung der Anforderung A2 nach der Bereitstellung der BpmService-Schnittstelle.

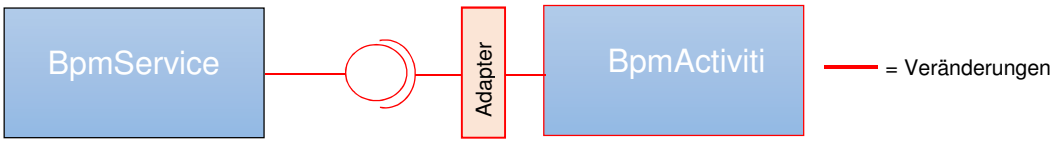


Abbildung 17: Komponentenbeziehung – Erstellung Activiti-Komponente

[59] *Legende: [+] = Umsetzbar, jedoch Anpassungen notwendig; [o]: Out of the Box nutzbar*

Activiti bietet die Möglichkeit eigene Funktionalität zu hinterlegen, welche beim Erreichen eines User Tasks ausgeführt wird. Es ist notwendig dafür einen eigenen Softwarebaustein bereitzustellen, welcher die von Activiti dafür vorgesehenen Schnittstellen nutzt und gegenüber der Postkorb-Komponente als Brücke fungiert (Anforderung A3). Ebenfalls muss diese Erweiterung den Rückkanal vom Postkorb zu Activiti abbilden, um Anforderung A6 zu erfüllen.

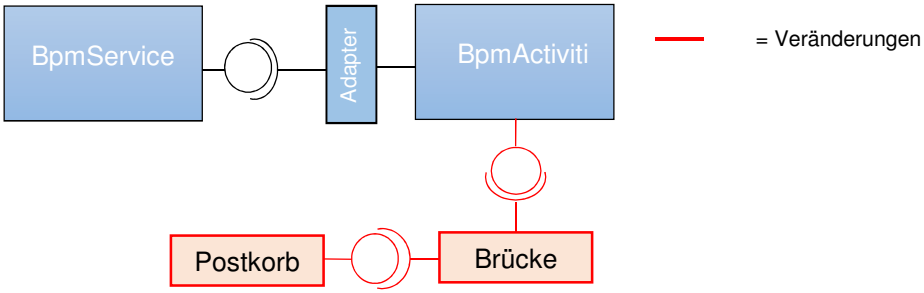


Abbildung 18: Komponentenbeziehung – Änderung der Engine-Funktionen

5.4 Zusammenfassung

Aus den im vorangegangenen Kapitel aufgestellten Anforderungen wurde ein Lösungskonzept abgeleitet. Dies umfasst verschiedene Softwarebausteine sowie Schnittstellen, welche Integration von Activiti unter den gegebenen Forderungen ermöglicht. Die Realisierung dieses Konzeptes wird im folgenden Kapitel gezeigt.

6. Integration

Die Umsetzung des Konzeptes erfolgt am Beispiel des Geschäftsprozess Vertragsanbahnung. Als Systemumgebung dient eine Oracle Datenbank 11g Enterprise Edition, einem Eclipse Indigo Entwicklungsumgebung sowie der Activiti Engine 5.13 und den Activiti Designer 5.12.

6.1 Prozess Vertragsanbahnung

Die Vertragsanbahnung dient der Erstellung eines Leasingvertrages vom Beginn des Antrageingangs, bis hin zur Aktivierdung des Vertrages. Während der Prozessausführung haben die Bediener mehrere Aufgaben zu erledigen. Die Aufgaben werden abteilungsübergreifend vergeben und über das HWI bedienerspezifisch ausgeführt. Der aktuelle Verlauf des Vertragsanbahnungsprozess wird über den Source Code abgebildet. Der zugrundeliegende Geschäftsprozess ist in Anhang H[2] visualisiert. Die in Abbildung 19 dargestellten Use-Cases basieren alle auf Nutzereingaben (User Task). Die Kommunikation zwischen den beteiligten Akteuren erfolgt über das Postkorb-System. Weitere Strukturmodelle sind im Anhang H aufgeführt.

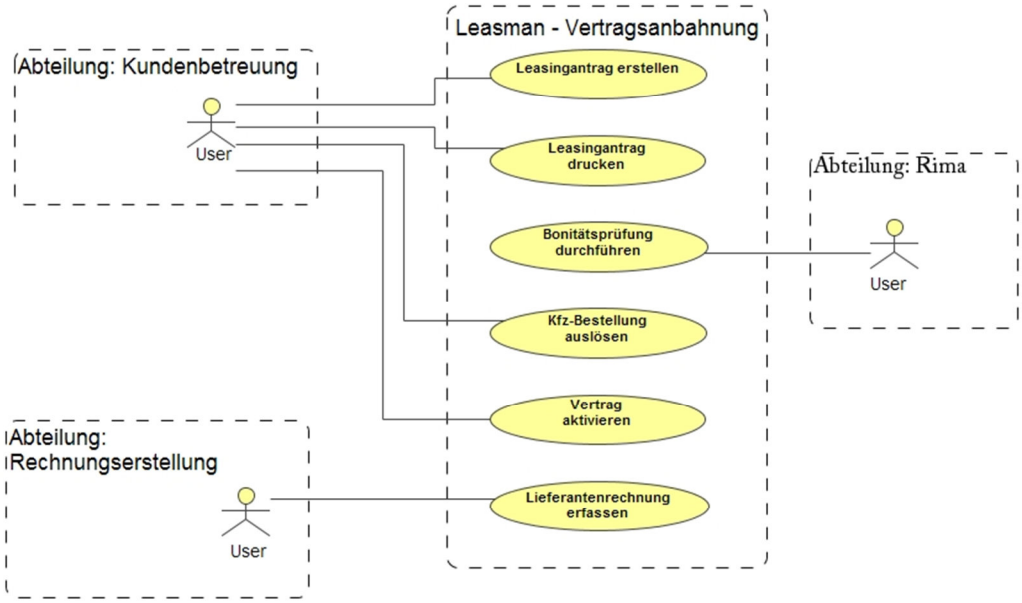


Abbildung 19: Use-Case Diagramm Vertragsanbahnung

6.2 Erstellung der Activiti Komponente

Für die Erstellung werden zunächst die Libraries von Activiti in Eclipse importiert und dem für Activiti erstelltem Package bekannt gemacht. Anschließen erfolgt, nach den im Kapitel 4.3.1 beschriebenen Regeln, die Programmierung der Interfaces IBpmAvtivitiService und IBpmactivitiManager, für den Zugriff externer Anwendungen und interner Komponenten.

Bei der Erstellung des Entwurfsmusters wurde beschlossen, dass diese Komponente ein Fragment wird, d.h. diese Komponente wird andere Komponenten unterstützen, selbst aber von keiner anderen Komponente benutzt. Danach Beginnt die Programmierung der Logik, wie das Starten oder Fortführen von Prozessen. Diese ist in der Klasse BpmActivitiManager enthalten.

6.2.1 Initialisieren der Engine

In Kapitel 4 wurde erklärt, dass der Leasman das Spring-Framework unterstützt und darüber seine Komponenten startet. Spring wird in Activiti als eine Möglichkeit genutzt, um die Prozess-Engine zu starten. Die zweite Möglichkeit erfolgt in einer Java-Klasse.

Das Starten der Engine über eine Klasse ist komplexer und erzeugt mehr Kopplungen zwischen den Komponenten. Die Komplexität des Leasman muss gering wie möglich gehalten werden, weswegen Spring für das Hochfahren der Engine genutzt wird.

Des Weiteren wurde in Kapitel 4 beschrieben, dass nur Komponenten über Spring, durch das BLS erstellt werden. Da Activiti ein Fragment ist und keine anderen Komponenten auf sie zugreifen dürfen, kann die Prozess-Engine dort nicht starten. Für dieses Problem gibt es zwei Alternativen. Zum einen kann das Starten im Superinterface erfolgen oder zum anderen in gesonderten Komponenten.

Da BPM-Engines genutzt werden, um alle Varianten von Prozessen abzubilden, darunter auch Prozesse, die keine Datenhaltung benötigen sowie weil die Übersichtlichkeit der Leasman-Architektur erhalten bleiben soll und die Tatsache, dass die Kunden der DELTA proveris AG nicht alle Initialisierungsoptionen benötigen wurde sich für die Variante von zusätzlichen Komponenten entschieden.

Diese Komponenten enthalten nur die Konfigurations-XML-Dateien für die BPM-Engines, mit und ohne Datenbank. Sie bekommen Zugriff auf den BpmService.

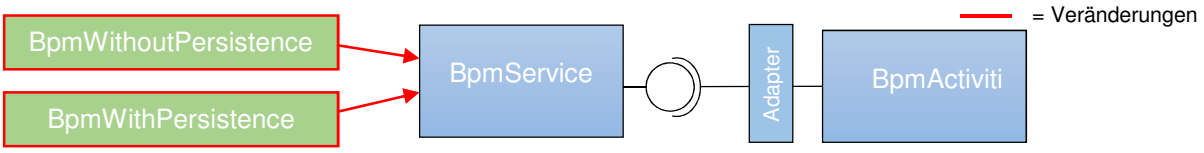


Abbildung 20: Komponentenbeziehung – BPM-Hochfahrkomponenten

Dieser Abschnitt erfüllt Anforderung A1 und Anforderung A2.

6.2.2 Konfiguration der Engine

Innerhalb der Konfigurations-Dateien werden Beans mit den Einstellungen der Prozess-Engine und damit deren Starten definiert. Rademarks ([RAD12] S.178f) beschreiben kurz, welche Faktoren bei der Standardkonfiguration einer Activiti-Engine wichtig sind. Der Funktionsumfang, den diese Faktoren dabei haben, ist groß, weshalb eine detaillierte Beschreibung bei [RAD12] fehlt. [AUG13] deckt ein breiteres Spektrum an Funktionen ab.

Die Konfigurationsdatei für die Einstellungen der Activiti-Engine ist im Spring Format aufgebaut. In dieser XML-Datei sind 4 Beans enthalten, die definitiv für das Hochfahren und die Nutzung der Engine vorhanden sein müssen.

Da es sich bei Konfigurationsdatei um eine XML-Datei handelt, muss zunächst die richtige Auswahl der Namespaces getroffen werden:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-2.5.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">
```

Die erste relevante Bean ist die Datenbank auf die Activiti zugreift. Der Leasman beinhaltet solch eine Bean bereits, welche bei er Vertragsanbahnung zum Einsatz kommt. Die Konfiguration einer Datenbank sieht wie folgt aus:

```
<bean id=“example.dataSource“
class=“org.springframework.jdbc.datasource.SimpleDriverDataSource“>
       <property name=“driverClassName“ value=“oracle.jdbc.driver.OracleDriver“/>
       <property name=“url“ value=“jdbc:oracle:thin:@localhost:1521:xe“/>
       <property name=“username“ value=“example“/>
       <property name=“password“ value=“example“/>
</bean>
```

Bei jBPM 4.3 gab es die Möglichkeit, über eine Methode die Suche nach einer Datenbank zu unterbinden. Activiti bietet diese Möglichkeit nicht an oder sie wurde während der Forschungsarbeit nicht entdeckt.

Als Alternative bietet Activiti die Variante einer „In Memory“-Datenbank. Mit dieser Einstellung wird eine Datenbank erschaffen, welche nach dem Abschluss der Prozessinstanz wieder geschlossen wird und damit alle gespeicherten Informationen verloren gehen. Eine Datenbank dieser Art könnte wie folgt aussehen:

```
<bean id=“withoutpersistence.dataSource“
class=“org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy“>
        <property name=“targetDataSource“>
                <bean class=“org.springframework.jdbc.datasource.SimpleDriverDataSource“>
                        <property name=“driverClassName“ value=“org.h2.Driver“/>
                                <property name=“url“ value=“jdbc:h2:mem:activiti;
                                DB_CLOSE_DELAY=864000“/>
                        <property name=“username“ value=““/>
                        <property name=“password“ value=““/>
                </bean>
        </property>
</bean>
```

Weitere Einstellungsmöglichkeiten sowie die Datenbanken die Activiti unterstützt, sind im Anhang G[1] aufgeführt.

Nachdem die Datenbank als Bean erstellt wurde, benötigt die Prozess-Engine noch eine Bean, die sich um die Transaktionen zwischen der Prozessdefinition oder Java-Klassen und der Datenbank kümmert. Auch hierbei ist eine Bean bereits im Leasman enthalten, die nur in der Prozess-Engine-Einstellungsbean eingebunden wird.

```
<bean id="txHibernateManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="{hier die ID der Session Factory-Bean angeben}"/>
</bean>
```

Allgemein und für die Konfigurations-XML des WithoutPersistence-Package wird dafür folgende Bean-Struktur gewählt:

```
<bean id="example.transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="example.dataSource"/>
        <!-- optional -->
        <property name="sessionFactory" ref="{hier die ID der Session Factory-Bean angeben}"/>
</bean>
```

Die Eigenschaften der Prozess-Engine sind auf zwei Arten realisierbar. Zum einen besteht die Möglichkeit die Einstellungen in einer Klasse zu definieren und als zweites können in einer XML-Datei alle Einstellungen vorgenommen werden.

Der Unterschied dabei liegt beim Aufbau der Bean. Der Quelltext für die Erstellung der Bean über eine Java-Klasse kann wie folgt aussehen:

```
<bean id="example.processEngineConfiguration" class="{Packagename}.{Klassenname}">
        <!-- optional sind hier Eigenschaften definierbar oder in der Klasse selbst -->
</bean>
```

Für die Vertragsanbahnung ist in einer XML folgende Bean relevant:

```
<bean id="withPersistence.processEngineConfiguration"
class="org.activiti.spring.SpringProcessEngineConfiguration">
        <property name =“databaseType“ value=“oracle“>
        <property name="dataSource" ref="MandantDataSource"/>
        <property name=“transactionManager“ ref=“txHibernateManager“/>
        <property name=“history“ value=“full“
</bean>
```

Weitere Einstellungsmöglichkeiten werden im Anhang G[1]. aufgezeigt und erklärt.

Die Prozess-Engine selbst braucht auch eine Bean, der nur die vorher erzeugten Einstellungen für die Engine übergeben werden:

```
<bean id="withPersistence.processEngine" class="
org.activiti.spring.ProcessEngineFactoryBean">
        <property name =“processEngineConfiguration“ ref=“withPersistence.
        processEngineConfiguration“/>
</bean>
```

Ein Tag, das mit in der Konfigurationsdatei enthalten sein könnte, sorgt dafür, dass in die Transaktionen des Transaktionsmanager eingegriffen werden kann.

```
<tx:annotation-driven transaction-manager=“txHibernateManager“/>
```

Im Anhang G sind weitere Beans und Properties aufgeführt. [AUG13] gibt einen detaillierteren Überblick über die Konfigurationsmöglichkeiten der Activiti Engine.

6.2.3 Erweiterung des Standard-Engine-Source Code

In Activiti werden die User Tasks als Aufgaben angesehen. Da der Leasman eigene Aufgaben besitzt, muss der Entwickler sich überlegen, wo und wie die Leasman-Aufgaben erstellt werden. Zum einen kann er den Source Code von Activiti erweitern oder er benutzt die Listener von Activiti. Da die Pflege des Source Codes beim Eingriff in den Activiti Source Code einfacher ist, wurde für die Vertragsanbahnung entschieden, dass beim Erreichen eines User Task die Aufgabe für den entsprechenden User oder Gruppe angelegt wird. Die Möglichkeit über Listener wird in Kapitel 6.5.1 vorgestellt. Um diese Änderungen vorzunehmen muss, zunächst ein Blick in die activit-engine.jar geworfen werden. Je nach Eingriff müssen die entsprechenden Klassen aus dieser Library geändert werden.

Für die Änderungen in der Vertragsanbahnung ist die erste relevante Klasse die FlowNodeActivityBehavior. Aus dieser Klasse bilden sich die Spezialisierungen der einzelnen Events (Bsp. UserTaskActivityBehavior). Die Spezialisierung beinhalten die Methoden zur Ausführung (execute) und Fortsetzung (signal) eines Events. Um nun die Methoden dieses Events zu überarbeiten, muss die entsprechende Spezialisierung erweitert (extends) werden.

```
public class UserTasksActivityBehavior extends UserTaskActivityBehavior implements
IBpmCommonDefaults{
	public BpmUserTasksActivityBehavior(TaskDefinition p_taskDefinition) {
		super(p_taskDefinition);
	}

	@Override
	public void execute(ActivityExecution p_execution) throws Exception {
		Map<String, Object> taskVariables = createTaskVariables(p_execution);
		p_execution.setVariables(taskVariables);
		m_managePostkorbAufgabe.createNewPostkorbAufgabe(taskVariables);
		super.execute(p_execution);
	} […]
}
```

Die zweite relevante Klasse ist der AbstractFlowNodeBpmnParseHandler. Die Spezialisierungen dieser Klasse (Bsp. UserTaskParseHandler) sind ebenfalls eventspezifisch. Sie beobachten den Prozess und falls das entsprechende Event erreicht wird, starten diese Handler die Spezialisierungen des FlowNodeActivityBehavior. Das bedeutet auch, der entsprechende Handler eines Events muss überschrieben (extends) werden.

```
public class CustomUserTaskBpmnParseHandler extends UserTaskParseHandler{
	@Override
	protected void executeParse(BpmnParse, UserTask userTask) {
		// Übernahme des 1.Teil aus der UserTaskParseHandler
		[…]
		// Veränderung des letzten Abschnittes durch Bekanntgabe der eigenen
		// UserTaskBehavior
		UserTasksActivityBehavior utab = new UserTasksActivityBehavior(taskDefinition);
		activity.setActivityBehavior(utab);
	}
}
```

Die Engine selbst muss nun noch über den neuen Handler Bescheid wissen. Dazu findet eine Modifizierung der Prozess-Engine-Einstellungsbean statt. Damit ist Anforderung A3 bestätigt.

```
<bean id=" withPersistence.processEngineConfiguration "
class="org.activiti.spring.SpringProcessEngineConfiguration">
        […]
        <property name=“customDefaultBpmnParseHandlers“>
                <list>
                        <!--Liste der zu nutzenden Handler innerhalb der Engine -->
                        <bean class=“CustomUserTaskBpmnParseHandler“ />
                </list>
        </property>
</bean>
```

Dazu wird das Attribut customDefaultBpmnParseHandlers hinzugefügt. Mit diesem Attribut wird die Liste der Handler um die eigenen Handler erweitert.

6.2.4 Modell der Vertragsanbahnung

Im Kapitel 2.3 wurde bereits auf die Begriffe Prozessdefinition und –diagramm eingegangen. Das Prozessdiagramm ist dem zufolge das reine gezeichnete Model ohne Logik, die Prozessdefinition ein ausführbares Prozessdiagramm, d.h. die Events und der Prozess beinhalten in ihren Eigenschaften ausführbare Logik. Dazu kommt noch die Prozessinstanz, welche eine ausgeführte Prozessdefinition ist. Es können gleichzeitig mehrere Prozessinstanzen derselben Prozessdefinition existieren (1:n - Beziehung).

Zur Erstellung des Prozessdiagramms muss erneut das Vertragsanbahnungskonzept analysiert werden. Der Prozess beginnt, nachdem ein Angebot eingegangen ist und dazu die entsprechende Aufgabe im Postkorb eines Users angelegt wird. Je nach Antragsstatus (eine sich ändernde Prozessvariable) wird die Antragsbearbeitung fortgesetzt oder beendet. Bei Fortsetzung des Antrags wird der Wert des Status geprüft und der nächste Pfad entsprechend gewählt.

Beim Erreichen des nächsten Tasks wird definitiv wieder eine Aufgabe an den zuständigen Bediener, Gruppe oder Abteilung erstellt. Nach diesem Schema findet der komplette Prozess statt, da außer Usertasks keine anderen Tasks vorhanden sind.

Mithilfe des Konzepts und dem erstellten Aktivitätsdiagramm kann anschließend der Prozess im Activiti Designer erstellt werden. Über die Drag & Drop Funktion des Activiti Designers wird der Prozess modelliert und gespeichert.

Um das Activiti Diagramm mit der Activiti-Engine zu nutzen, muss die „*.bpmn20-Datei“ in eine XML-Datei konvertiert werden. Diese erfolgt durch die Änderung des Dateiformats. Das bedeutet aus „*.bpmn20“ wird „*.bpmn20.xml“.

Der Vertragsanbahnungsprozess hängt von Benutzereingaben ab, die nicht sofort, d.h. während der Laufzeit des BLS, ausgeführt werden müssen. Dies führt dazu, dass diese Komponente über die BpmWithPersistence auf die BpmService zugreift.

Die Engine wird aber nicht nur genutzt, um Prozesse abzubilden, die immer eine Userinteraktion benötigen. Eine solche Komponente ist der ControlService. Der Prozess dieser Komponente besteht nur aus Service Tasks. Normalerweise läuft die Engine die Sequenz-Flows des Prozess solange ab, bis sie auf ein Event trifft, bei dem sie warten muss (z.B. User Task, Parallel Gateway, Timer Intermediate Event). Bei einem Service Task ist der Befehl des Wartens standardmäßig nicht gegeben, weswegen der Task ohne weitere Eingriffe ausgeführt und beendet wird.

Der Prozess des ControlService läuft ohne Unterbrechungen ab. Da momentan weder die Monitoring- noch die Reporting-Funktion von belangen sind, ist bei solchen Prozessen keine Datenhaltung notwendig. Der ControlService greift deshalb über die BpmWithoutPersistence auf den BpmService zu.

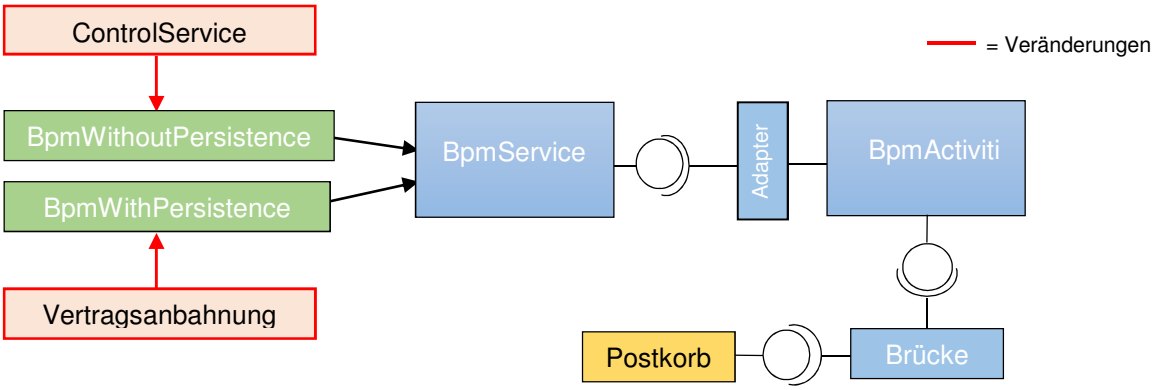


Abbildung 21: Komponentenbeziehung – Vertragsanbahnung & ControlService

6.3 Integration der Datenbank

6.3.1 Einbinden der Activiti-Tabellen

Activiti bietet die Möglichkeit die Leasman-Datenbanktabellen zu benutzen und an der entsprechenden Schnittstelle anzubringen. Die erfolgreiche Integration der Activiti Engine ist als Schwerpunkt angesehen, weshalb zunächst nur die Activiti Tabellen in die Datenbank des Leasman integriert werden.

Activiti bietet entsprechende Schnittstellen, wie die „{Datenbank-Relation} EntityManager.class“, an, um die Datenbanktabellen zu integrieren. Im Rahmen einer weiteren Arbeit ist die Möglichkeit der Integration der eigenen Tabellen näher zu betrachten.

Zur Integration der Activiti Tabellen in die eigene Datenbank (DB), werden die mitgelieferten SQL-Scrips aus „**activiti-engine.jar\org\activiti\db\create\activiti.{eigene DB}.create.*.sql“** ausgeführt. Activiti unterstützt sechs Datenbankenmanagementsysteme. Darunter auch Oracle, was ein weiterer Grund für die Auswahl von Activiti war.

Die Integration der Activiti Tabellen in die Leasman Datenbank hat zur Folge, dass sowohl Transaktionen vom Prozess zu den Activiti Tabellen stattfinden, als auch Transaktionen vom Prozess zu den Leasman Tabellen. Diese gemeinsame Transaktionsverwaltung muss konsistent ablaufen. Das bedeutet, dass Änderungen an den Geschäftsobjekten und dem Prozess korrekt in der Datenbank gespeichert werden, um Prozessstörungen durch falsche Datenhaltung zu vermeiden. Anforderung A5 ist mit diesem Abschnitt bestätigt.

6.3.2 Der Transaktionsmanager

> *Der Transaktionsmanager ist ein Teil einer Applikation, welcher für die Transaktionen der Anwendung mit einer Datenbank, zuständig ist. Bei fehlerfreier Ausführung speichert der Transaktionsmanager die Änderung in der Datenbank (commit), sonst setzt er die Datenbank zurück (rollback).*[60]

Der Leasman setzt auf das ORM „Hibernate“, um seine Transaktionen zu verwalten, während Activiti „iBatis“ einsetzt. Es ist zu untersuchen, ob Hibernate die Transaktionen, die für den Vertragsanbahnungsprozess anfallen, durchführt.

Dieses Problem kann bei jeder nachhaltigen Integration einer BPM-Engine in eine Geschäftsanwendung auftauchen und kann nur über wenige Möglichkeiten geklärt werden. Die Dokumentation ist eine Variante, um zu klären, welche ORMs unterstützt werden. Des Weiteren sind Foren oder Support sehr hilfreich. Die letzte Variante ist das Testen, ob benötigte Transaktionen stattfinden. Es ist nicht eindeutig dokumentiert, ob Hibernate nutzbar ist, weshalb auf Basis von Tests dessen Nutzbarkeit bestätigt wurde (siehe Anhang J).

[60] [WAL11] (Walls, 2011, S. 150 f)

Neben der Möglichkeit Hibernate in Activiti zu gebrauchen, kann auch überprüft werden, ob die Möglichkeit besteht zwei ORMs parallel zu benutzen. Dieser Abschnitt zeigt die Erfüllung von Anforderung A4.

6.4 Verbindung zum Human Workflow Interface

In Abbildung 16 wurde bereits gezeigt, dass der Postkorb das HWI für die BPM-Komponenten sein wird. Der Postkorb (vgl. [ZAH13] S.7) zeigt in erster Linie die Aufgaben an, die der User zu erledigen hat.

Das Postkorb Plug-In (Oberfläche Anhang A), der Leasman Workbench, führt die benötigten Methoden der Postkorbkomponente aus und zeigt das Ergebnis auf der Oberfläche (siehe Anhang A) an. Um dieses Plug-In zu nutzen, greifen sowohl die BpmService- als auch die Vertragsanbahnungskomponente auf den Postkorb zu.

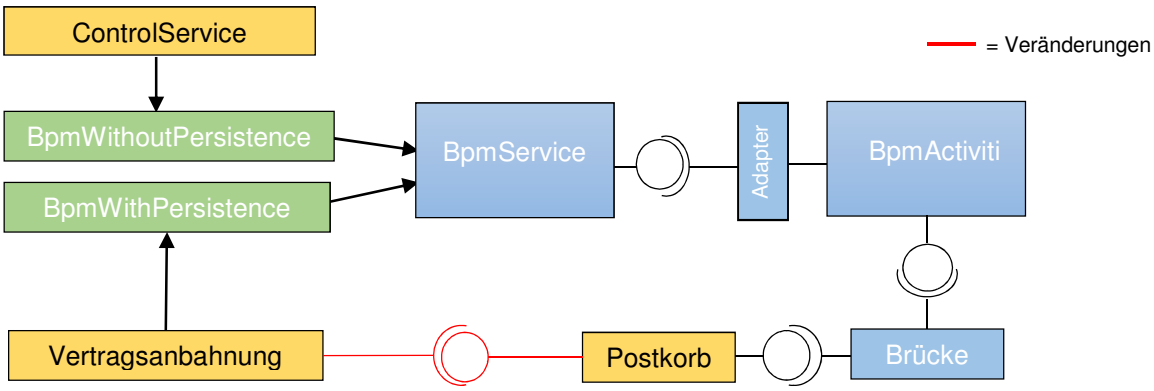


Abbildung 22: Komponentenbeziehung – Beziehung zum Postkorb

Anforderung A6 wird durch diesen Abschnitt umgesetzt.

6.5 Ausgewählte Funktionen von Activiti

6.5.1 Listener

Neben der Möglichkeit, eine Aufgabe über die Erweiterung des Engine-Source Codes zu erstellen, sind die Listener von Activiti eine Alternative. Die Listener beobachten die Prozessinstanz und beim Eintreten bestimmter Events werden diese aktiv. Dabei wird unterschieden zwischen dem Execution-Listener und dem Task-Listener.

Der Execution-Listener kann bei allen Events sowie bei der Prozessdefinition selbst angewandt werden. Er unterscheidet zwei wesentliche Events; bei denen er aktiv wird. Diese sind das Start- und End Event.

Das Erzeugen eines Execution-Listener erfolgt über den Activiti Designer oder manuell in der *.bpmn20.xml. Der Quellcode in der XML-Datei sieht wie folgt aus:

```
<userTask id="beispielUserTask" name="User Task testen">
        <extensionElements>
                <activiti:executionListener event="start"
                class="example.Listener.UserTaskExecutionListener"/>
                <activiti:executionListener event="end" delegateExpression="${
                userTaskExecutionListener }"/>
                <activiti:executionListener event="end"
                expression="${userTaskExecutionListener.execution(execution)}"/>
        </extensionElements>
</userTask>
```

Wie im Ausschnitt des Quelltextes gezeigt, gibt es 3 Möglichkeiten, den Execution-Listener zu benutzen. Über die „class“ und die „delegateExpression“ wird eine Klasse direkt aufgerufen. Dabei ist bei der delegateExpression darauf zu achten, dass eine entsprechende Bean zu der Klasse vorhanden ist. Dies kann in der Konfigurations-XML oder einer anderen initialisierten XML-Datei erfolgen:

```
<bean id=" userTaskExecutionListener" class="
example.antragsbearbeitung.process.Listener.UserTaskExecutionListener "/>
```

Bei der Nutzung der expression-Variante ist ebenfalls darauf zu achten, dass eine entsprechende Bean vorhanden ist. Die expression-Variante wird allerdings benutzt, um bestimmte Methoden von Klassen aufzurufen. Im Gegensatz zu den anderen beiden Varianten wird in dieser Variante keine Methode der Activiti-Library benötigt.

Bei den beiden ersten Varianten muss die aufzurufende Klasse den „ExecutionListener“ implementieren. Dieser beinhaltet die Methode „notify()“ die automatisch beim Starten der Klasse über die Prozessdefinition ausgeführt wird.

```
public class UserTaskExecutionListener implements ExecutionListener{
        private static final long serialVersionUID = 1L;
        @Override
        public void notify(DelegateExecution execution) throws Exception {
                execution (execution);
        }

        public void execution(DelegateExecution execution){
                createNewAufgabe(execution);
        }
[…]
}
```

Der Task-Listener beinhaltet ebenfalls die drei Varianten des Klassen- & Methodenaufrufs: class, delegateExpression und expression. Allerdings wird dieser standardmäßig nur beim User Task eingesetzt. Er beobachtet den User Task auf die Events „create“, „complete“ und „assignment“.

Innerhalb der Klassen liegt der Unterschied zum Execution-Listener darin, dass der „TaskListener“ implementiert wird. Auch dieser enthält die „notify()“-Methode.

In der Vertragsanbahnung werden kein Listener genutzt. Vollständigkeitshalber zeigt Anhang I nochmal die Reihenfolge der Execution- und Task-Listener-Events.

6.5.2 Exclusive Gateway

Innerhalb der Vertragsanbahnung sind einige „Entweder-Oder“ Situationen, die durch ein Exclusive Gateway dargestellt werden. Die Entscheidung, welcher Fluss gewählt wird, kann auf mehrere Arten erfolgen. Zum einen kann der Sequenz-Flow die Prozessvariablen analysieren und je nach Wert den entsprechenden Pfad wählen:

```
[…]
<exclusiveGateway id="entscheidungUser" name="User Entscheidung"></exclusiveGateway>
<sequenceFlow id="ablehnung" name="Ablehnung" sourceRef="entscheidungUser"
targetRef="druckAblehnung">
        <conditionExpression xsi:type="tFormalExpression"><![CDATA[${status=='ablehnen'}]]/>
</sequenceFlow>
<sequenceFlow id="zustimmung" name="Zustimmung" sourceRef="entscheidungUser"
targetRef="exclusivegateway2">
        conditionExpression xsi:type="tFormalExpression"><![CDATA[${status=='zustimmen'}]]/>
</sequenceFlow>
[...]
```

Ebenfalls kann ein Execution-Listener genutzt werden. Um dem Exclusive Gateway einen Execution-Listener bekannt zu machen, gibt es nur die Möglichkeit über die *.bpmn20.xml-Datei. Der Activiti Designer bietet diese Möglichkeit nicht, er zeigt allerdings den selbst erstellten Execution-Listener in den Prozessdefinitions-eigenschaften an.

Probleme bei dieser Variante treten auf, falls Änderungen über den Activiti Designer erfolgen. Das Execution-Listener-Tag aus der *.bpmn20.xml-Datei wird in dem Fall vom Exclusive Gateway-Tag entfernt und in das Prozessdefinition-Tag gesetzt. Bei der Nutzung dieser Möglichkeit muss der ExecutionListener und die ActivityBehavior implementiert werden. Hier sei zum ActivityBehavior nur gesagt, dass diese Klasse es ermöglicht den Sequenzflow selbst zu bestimmen. Mehr zum ActivityBehavior wird in Kapitel 6.5.4 berichtet.

```
public class ExclusivGatewayListener implements ExecutionListener, ActivityBehavior {
	@Override
	public void execute(ActivityExecution execution) throws Exception {
		PvmTransition transition = null;
		if (execution.getVariable("status").equals("zustimmen")) {
			transition = execution.getActivity().findOutgoingTransition("zustimmung");
		} else {  transition = execution.getActivity().findOutgoingTransition("ablehnung");}
		execution.take(transition);
	}
	@Override
	public void notify(DelegateExecution execution) throws Exception {
		execute((ActivityExecution) execution);
	}
}
```

6.5.3 Umgang mit Exception

Innerhalb eines Prozesses können unerwarteten und erwarteten Fehlern auftreten. Erwartete Fehler können durch den Einsatz von Error-Boundary Events behandelt werden. Dazu wird dieses Event an den entsprechenden Task angeheftet, ein Error-Tag erstellt und die Klasse in dem es zu den Fehler kommt, mit dem entsprechenden Code versehen. Als Beispiel sieht der Code für die *.bpmx20-Datei wie folgt aus:

```
<definition […]
	<error id=" exampleMistake " errorCode="0"></error>
	<process […]
		<serviceTask id="fehlerExample" name="Example fehler"
		activiti:class="example.antragsbearbeitung.antrag.Antrag"> […]
		</serviceTask>
		<boundaryEvent id="boundaryerror1" name="Error" attachedToRef="
		fehlerExample ">
			<errorEventDefinition errorRef="exampleMistake"></errorEventDefinition>
		</boundaryEvent>
	</process>
</definition>
```

Dies ist aber nicht die einzige Variante. Die zweite Variante ist dieselbe, wie bei unerwarteten Fehlern. Es wird innerhalb der Klasse der Fehler abgefangen und dann mittels Java-Code eine Fehlerbehandlung durchgeführt.

```
public class Antrag implements JavaDelegate{
[…]
@Override
	public void execute(DelegateExecution execution) throws Exception {
		try{
			validatAntrag(Datenbankabfragen.findAntragById((int)
			execution.getVariable("antragId")));
		} catch (ActivitiException e){
}			throw new BpmnError(„validierungFehlgeschlagen“);
		}
[…]
}
```

6.5.4 Interaktion mit Service Task

Der Service Task ermöglicht den Aufruf einer Klasse oder das Nutzen einer Expression und Delegate Expression. Die Expression und die Delegate Expression funktionieren wie beim Execution-Listener (siehe Kapitel 6.5.1).

Beim Aufruf einer solchen Klasse bietet Activiti die Möglichkeit einer Feldeinkopplung (field injection). Das bedeutet die Activiti-Engine liest, während der Bearbeitung des Service Task, alle Task-Eigenschaften, die in der Prozessdefinition definiert wurden, aus und übergibt die Werte in die angegebene Klasse. Diese Eigenschaften werden mit dem Tag <activiti:field> angegeben. Die Möglichkeit, Eigenschaften in der Prozessdefinition einzubringen und diese auszulesen, bieten Events mit der „class“-Anweisung (Service Task und die Listener).[61]

```
<serviceTask id="antragValidieren" name="Antrag validieren"
activiti:class="example.Vertragsanbahnung">
        <extensionElements>
                <activiti:field name="vertragId">
                        <activiti:expression>${vertragId}</activiti:expression>
                </activiti:field>
        </extensionElements>
</serviceTask>
```

Damit die Zuordnung der Werte funktioniert, muss innerhalb der angegebenen Klasse ein Attribut mit demselben Namen wie der Name des <activiti:field> angelegt werden und mit dem Datentyp „Expression“ existieren.

```
private Expression vertragId;
```

Nachdem die Engine die Klasse aufgerufen und die Methoden ausgeführt hat, muss ihr noch mitgeteilt werden, welchen Pfad sie zur Prozessfortsetzung verfolgen muss. Dazu wurden bereits zwei Methoden vorgestellt, die nun näher erläutert werden.

Eine Möglichkeit ist die Implementierung der ActivityBehavior Klasse (Quelltext Kapitel 6.5.2). Diese Klasse sollte nur Verwendung finden, wenn es nötig ist, selbst zu bestimmen, welcher Ausgangspfad verwendet wird. Bei Implementierung dieser Klasse ist diese Angabe Pflicht. Es lohnt sich also nur der Einsatz, wenn der Service Task wie ein Inclusive Gateway fungieren kann.
Normalerweise reicht die Implementierung der zweiten Variante, die JavaDelegate (Quelltext Kapitel 6.5.3), aus. Sie besitzt ebenfalls die execute()-Methode, sucht sich aber den Ausgangspfad selbst.

[61] [AUG13] (Activiti - User Guide, 09.08.2013 - 10:14)

6.5.5 Variablen

Variablen sind prozessspezifische Objekte, die während der Ausführung des Prozesses innerhalb der *.bpmn20-Datei oder einer Klassen erstellt, aufgerufen und verwaltet werden.

```
public class Vertrag implements JavaDelegate{
[…]
        @Override
        public void execute(DelegateExecution execution) throws Exception {
                String antragId = execution.getVariable("antragId")));
                antragId = „01“;
                execution.setVariable(„antragId“, antragId);
        }
[…]
}
```

Es wird zwischen Prozess- und Taskvariablen unterschieden. Der Hauptunterschied zwischen den beiden Variablen liegt in deren Lebensdauer. Prozessvariablen sind während des kompletten Prozesses verfügbar, während Taskvariablen nur solange vorhanden sind, wie der Task existiert.

6.6 Zusammenfassung

Dieses Kapitel zeigte die durchgeführte Variante, für die Integration von Activiti in den Leasman. Sie ist eine von vielen Möglichkeiten, solch eine Integration durchzuführen.

Dabei wurden die Anforderungen des Konzeptes geprüft, erläutert und umgesetzt. Es wurde bestätigt, dass die Anforderungen A1 bis A6 erfolgreich umgesetzt sind und damit die Integration erfolgreich war.

Die beschriebenen Funktionen sind für die Vertragsanbahnung relevant, sollten aber vor jeglicher Nutzung eines BPMS untersucht werden, da so die Komplexität der Geschäftsanwendung erhöht oder gesenkt wird.

7. Abschlussbetrachtung

„BPM als Kombination aus Prozessmanagement und Software ist die Lösung für die Suche nach permanenter Optimierung – im Prozess die richtigen Informationen, zur rechten Zeit, am rechten Ort zu haben, ist für unsere Kunden Gold wert.“ [62]

Dr. Florian Bergener
Value Engineer – Sales EMEA Central Region bei OpenText

7.1 Ergebnis der Arbeit

Ziel der vorliegenden Arbeit war es, die Integration einer BPM-Engine (Activiti) in eine komplexe Geschäftsanwendung (Leasman) zu realisieren. Um dies zu erreichen, mussten die Fragen aus Kapitel 1 untersucht und beantwortet werden. Diese waren:

- Welche Voraussetzungen bringt eine BPM-Engine mit, um als Embedded-Lösung eingesetzt zu werden?
- Welche Anforderungen stellt Leasman, damit eine Applikation integriert werden kann?
- Ist eine Anpassung der BPM-Engine an die Funktionen des Leasman nötig und wie findet dies gegeben falls statt?

Nachdem die relevanten BPM-Begriffe geklärt wurden, begann die Analyse der Anwendungen. Ein Vergleich der beiden Analysen zeigte, dass die Voraussetzungen einer erfolgreichen Integration gegeben waren.

Das aus den Analysen erstellte Integrationskonzept verdeutlichte die Anforderungen an die neue Komponente und zeigt welche Architekturbereiche des Leasman genauerer betrachtet werden müssten. Am Beispiel der Vertragsanbahnung wurde die Realisierung des Lösungskonzeptes gezeigt.

Anschließend wurde die Umsetzung des Konzeptes realisiert. Dabei wurde zum einen aufgezeigt, wie die Integration von statten geht und zum anderen musste der Source Code der BPM-Engine erweitert werden. Dabei wurde die BPM-Engine erfolgreich gestartet und die Erweiterungen im Source Code implementiert sowie getestet. Nebenbei wurden auch wichtige Funktionen einer BPM-Engine vorgestellt.

[62] [MES13] (Messe-Stuttgart - BPM, 15.08.2013 – 10:04)

Diese BPM-Komponente vereinfacht die Automatisierung von Geschäftsfunktionen und senkt die Komplexität, wodurch eine bessere Übersicht über den Quelltext

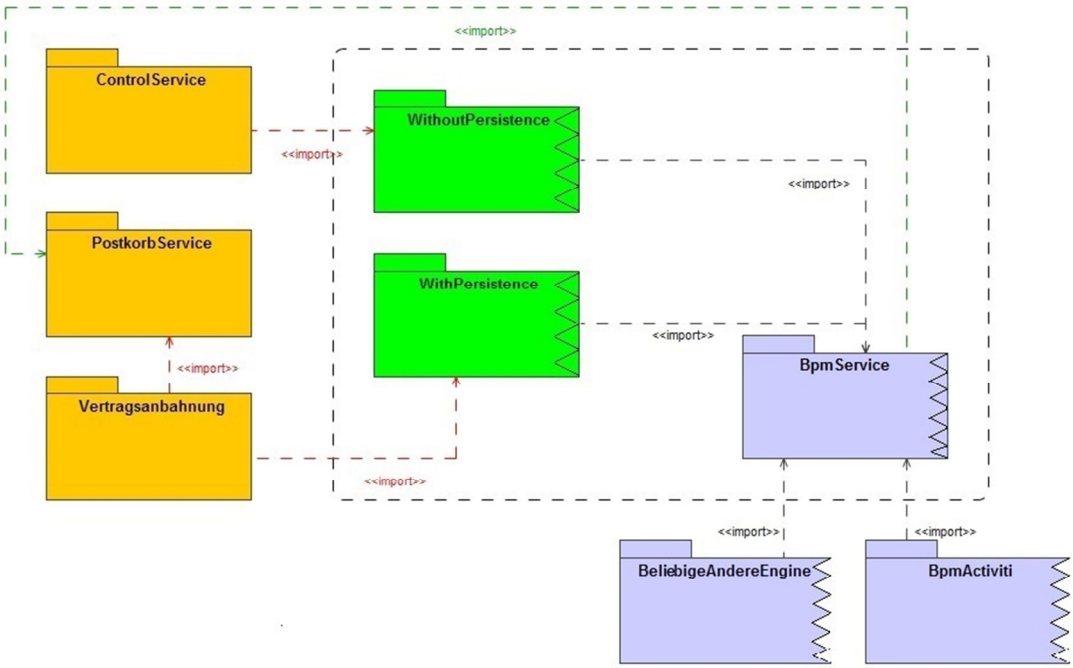


Abbildung 23: Packagediagramm vom BPM im Leasman

entsteht. Der Ist-Zustand wird durch das folgende Packagediagramm veranschaulicht.

7.2 Einschätzung des Lösungsverfahrens

Für ein effektives Vorgehen bei der Entwicklung der BPM-Komponente wurde zunächst die Softwarearchitektur des Leasman untersucht. Anhand vorhandener Dokumentationen und Erfahrung der Mitarbeiter stellte die Untersuchung keine Schwierigkeiten dar. Das dadurch gewonnene Wissen macht sich nicht nur bei der Bearbeitung dieser Arbeit bezahlt, sondern auch für zukünftige Entwicklungs- und Analyseaufgaben. Die daraus resultierenden Ergebnisse ermöglichten eine effiziente Darstellung mittels UML-Diagramm-Typen, wie Klassen- oder Aktivitätsdiagramm.

Des Weiteren war eine Analyse von Activiti nötig. Der angebotene User Guide ist zwar sehr umfangreich und das Forum deckt viele Fragen der User ab. Aber einige für die Vertragsanbahnung wichtige Funktionen sind nicht oder nicht ausführlich genug erklärt und musste durch Selbststudium in Erfahrung gebracht werden. Das Selbststudium fand anhand eines kleineren Beispiels statt.

Der Vergleich beider Anwendungen zeigte, in wie weit die Anforderungen der Geschäftsanwendung durch das BPMS erfüllt wurden und ob eine Integration möglich ist. Die Unterschiede, wie Datenbanktabellen und ORM, wurden durch die Festlegungen der DELTA proveris AG oder anhand des Selbststudiums gelöst.

Die in dieser Arbeit aufgezeigte Architektur, für das Einbinden einer BPM-Engine in eine komplexe Geschäftsanwendung, ist eine von vielen Möglichkeiten und soll vor allem die Austauschbarkeit der Engine ermöglichen.

7.3 Ausblick

Die Nutzung einer BPM-Engine in seiner eigenen Geschäftsanwendung ist sicher sehr reizvoll. Allerdings ist immer zu beachten, dass die technische Seite der eigenen Anwendung auch die Möglichkeit einer Anbindung bietet.

Der in dieser Arbeit geschilderte Weg der Integration lässt sich, aufgrund der Individualität, nicht auf jede Engine anwenden, bildet aber eine Anleitung wie die Durchführung einer Integration durchgeführt werden kann und was es dabei zu beachten gibt.

Neben den beschriebenen Funktionen werden im Folgenden weitere Funktionen vorgestellt, deren Einbindung sinnvoll ist, aber speziellen Untersuchungen unterliegen.

7.3.1 Monitoring und Reporting

Die Funktion Monitoring, oder auch Überwachung, ist für Geschäftsprozesse unabdingbar. Sie dient dazu Schwachpunkte im Prozess zu finden und diese durch geschicktes Bearbeiten zu lösen. Viele BPMS bieten solch eine Funktion an. Sie wird unter anderem in der Simulation oder während des Prozessverlaufes eingesetzt. Das Monitoring ist für jedes Unternehmen reizvoll und nach Bedingungen des § 14 Abs.1 und 2 BDSG zulässig. Allerdings ist die Speicherung personenbezogener Daten ein umstrittenes Thema. Deswegen muss sich das Unternehmen im Klaren sein, dass es aufgrund arbeitsrechtlicher Aspekte zu Problemen mit den Mitarbeitern kommen kann.

7.3.2 Datenbanktabellen

Es wurde bereits aufgeführt, dass die Activiti Datenbanktabellen nur in das bestehende Datenbanksystem übernommen wurden. Interessant wäre eine Analyse, in wie weit die Engine-Konfiguration bearbeitet werden muss, um seine eigenen Datenbanktabellen zu integrieren und wie sich der Umfang der Engine dadurch verändert. Ein Vorteil dabei wäre, dass eine doppelte Verwaltung der BPM-Daten wegfallen würde.

7.3.3 Eigene Business Process Management - Engine

Anhand der vielen Schnittstellen, die als Standard für eine BPM-Engine gefordert sind, besteht die Möglichkeit, dass eine eigene Engine auf Basis der Ausgangsengine geschaffen werden kann. Mit der eigenen Engine ist es möglich, sich weiter zu entwickeln sowie sich in neuen Bereichen des Softwaremarktes zu bewegen. Camunda produzierte ihr Tool Camunda Fox auf den Standards von Activiti.

Dieser Individualismus beherbergt allerdings das Risiko, dass Erweiterungen an der Ausgangsengine, in den Bereichen die vom Anwender geändert wurden, nicht möglich sind und Zeit und Kosten für Überarbeitungen anfallen.

Abkürzungsverzeichnis

Abkürzung	Begriff
BLS	Business Logic Server
BPM	Business Process Management
BPMN	Business Process Modeling and Notation
BPMS	Business Process Management System
CORBA	Comman Object Request Broker Architecture
DB	Datenbank
HWI	Human Workflow Interface
IS	Informationssystem
IT	Informationstechnik
JEE	Java Platform, Enterprise Edition
JPA	Java Persistence API
JTA	Java Transaction API
OMG	Object Management Group
OMS	Output Management System
ORM	Object-Relational-Mapping
RCP	Rich Client Platform
REST	Representational State Transfer
UML	Unified Modeling Language
WfMC	Workflow Management Coalition
WfMS	Workflowmanagement-System

Glossar[63]

Begriff	Erklärung
Ad-Hoc	aus dem Augenblick heraus, in diesem Moment
Handler	Funktion, die auf bestimmte Ereignisse spezialisiert und einem Objekt zugeteilt ist
Interface	Schnittstelle / Verbindungsstelle zwischen Funktionseinheiten eines Datenverarbeitungs- oder Datenübertragungssystems
Listener	Dient der Weitergabe von Änderungen an Objekte, die von dem beobachteten Objekt (Observer) abhängig sind
Notation	Ist ein System von Zeichen oder Symbolen einer Metasprache
Plug-In	kleines Softwareprogramm, das in eine größere Anwendung integriert werden kann
Release	Veröffentlichung (besonders einer neuen oder überarbeiteten Software)

[63] Die Erklärungen stammen aus [DUD13] und de.wikipedia.org

Literaturverzeichnis

[ACC13] Activiti-Komponenten. (28.07.2013). *Activiti*. Von Activiti: http://www.activiti.org/components.html abgerufen

[ACT13] Activiti-Unternehmen. (29.07.2013). *Activiti*. Von Activiti: www.activiti.org/team.html abgerufen

[AUG13] Activiti - User Guide. (05.08.2013). *Activiti*. Von Activiti: www.activiti.org/userguide/index.html abgerufen

[BAE13] Baeyens, T. (29.07.2013). *Blogspot*. Von Processdevelopments - Blogspot: http://processdevelopments.blogspot.de/2011/01/activiti-51-release-adds-activiti.html abgerufen

[BAR13] Bartonitz, D. M. (05.07.2013). *saperionblog*. Von saperionblog: http://www.saperionblog.com/wp-content/uploads/2011/11/Standards_BPM_2011_11.jpg abgerufen

[BAU07] Bauer, C., Gavin, K., & Dubau, J. (2007). *Java Persistence mit Hibernate .* München: Carl Hanser Verlag.

[CHI09] Chinnici, R., & Shannon, B. (2009). *Java™ Platform, Enterprise Edition (Java EE) Specification, v6.* Santa Clara: Sun Microsystems.

[CUL13] Cullman, X.-N., & Lambertz, K. (14.08.2013). *verifysoft.* (MScoder, Hrsg.) Von verifysoft: www.verifysoft.com/de_cmtpp_mscoder.pdf abgerufen

[DEA13] DELTA Proveris AG . (12.12.2013). Architektur Leasman. Limbach Oberfrohna, Sachsen, Deutschland.

[DEB12] DELTA proveris AG . (13.09.2012). Quo vadis - Leasman und Business Process Management. Limbach-Oberfrohna.

[DEL13] DELTA proveris AG - Leasman. (16.08.2013). *depag*. Von depag: htto://www.depag.de/leasman.html abgerufen

[DEM13] Deming, W. E. (12.09.2013). *Manager Wiki*. Von Manager Wiki: http://www.manager-wiki.com/unternehmensanalyse/84-business-process-management-bpm abgerufen

[DUD13] Bibliographisches-Institut-GmbH. (08.07.2013). *duden.de*. Von Duden: www.duden.de/rechtschreibung/ abgerufen

[EBE11] Ebert, R. (2011). *Eclipce RCP.*

[FEE13] Feess, P. D. (26. 08 2013). *Wirtschaftslexikon Gabler - Komplexität*. Von Wirtschaftslexikon Gabler: http://wirtschaftslexikon.gabler.de/Archiv/5074/komplexitaet-v8.html

abgerufen

[FOW03] Fowler, M. (2003). *Patterns of Enterprise Application Architecture.* Boston: Pearson Education, Inc.

[FRE10] Freund, J. (31. August 2010). Wie Sie im BPM-Dschungel eine passende Lösung finden. *Computerwoche*, S. 14-17.

[FUN10] Funk, B., Gómez , J. M., Niemeyer, P., & Teuteberg, F. (2010). *Geschäftsprozessintegration mit SAP.* Springer-Verlag.

[GAD10] Gadatsch, A. (2010). *Grundkurs Gechäftsprozessmanagement.* Wiesbaden: Vieweg+Teubner Verlag.

[GAR13] Gartner Inc. - BPM. (30.07.2013). *Gartner Inc.* Von Gartner Inc.: http://blogs.gartner.com/it-glossary/business-process-management-bpm-2/ abgerufen

[HAA06] Haag, D. (2006). Erstellung eines Architekturmodells für eine betriebl. Anwendung zur Organisation und Integration der Geschäftslogik in eine bestehnde Systemlandschaft auf Basis einer Geschäftsprozessanalyse und unter Berücksichtigung gegebener techni. Rahmenbedingungen. *Masterarbeit*. Mittweida: Hochschule Mittweida.

[HOH11] Hohwiller, J., & Dr. Schlegel, D. (2011). Erfolgreiche BPM Projekte durch ganzheitliche Betrachtung. *OBJEKTSpektrum*, S. 80-85.

[HOL95] Hollingsworth, D. (1995). *Workflow Management Coalition - The Workflow Reference Model.* Hampshire.

[HOR13] Horn, T. (28.08.2013). *torsten-horn*. Von torsten-horn: www.torsten-horn.de/techdocs/java-hibernate.htm#hbm.xml

[KOM11] Komus, A. (2011). *BPM Best Practice - Wie führende Unternehmen ihre.* Springer-Verlag.

[MES13] Messe-Stuttgart - BPM. (01.08.2013). *messe-stuttgart*. Von messe-stuttgart: http://www.messe-stuttgart.de/where-it-works/besucher/aktuelles/detailseite/btext/pac-veroeffentlicht-trendstudie-zu-bpm-in-der-dach-region/spm/1/socialmedia/1/backPid/25906/limit/10/an/showBusinesstextDetail/cl/Businesstext/ abgerufen

[MSD13] MSDN Microsoft - The Repository Pattern. (16.08.2013). *msdn*. Von msdn: msdn.microsoft.com/en-us/library/ff649690.aspx abgerufen

[MUL05] Müller, J. (2005). *Workflow-based Integration - Grundlagen, Technologien,*

Management. Springer Verlag.

[MUL11] Müller, J. (2011). *Strukturbasierte Verifikation von BPMN-Modellen.* Vieweg + Teubner Verlag.

[OMG13] OMG. (05.06.2013). *Über uns: OMG*. Von OMG.com: http://www.omg.org/gettingstarted/gettingstartedindex.htm abgerufen

[OSG13] OSGi - What Is OSGi. (26.08.2013). *OSGi*. Von OSGi: www.osgi.org/Technology/WhatIsOSGi

[OSI13] OSGi Alliance - About. (15.08.2013). OSGi Alliance. Von OSGi Alliance: www.osgi.org/About/HomePage?section=1 abgerufen

[POS13] Berliner BPM-Offensive - BPMN Poster. (30.07.2013). *BPM Offensive Berlin.* Von BPM Offensive Bwerlin: http://bpmb.de/images/BPMN2_0_Poster_DE.pdf - 05.07.2013 11:33 abgerufen

[RAD12] Rademakers, T. (2012). *Activiti in Action.* United States of America: Manning Publications Co.

[RIG09] Riggert, W. (2009). *ECM Enterprise Content Management.* Vieweg + Teubner | GEV Fachverlag GmbH.

[RIG13] Riggert, W. (14.08.2013). *FH-Flensburg.* Von FH-Flensburg: http://www2.wi.fh-flensburg.de/wi/riggert/veranstaltungen/AKAD/1-Integration.pdf abgerufen

[SCH98] Scheer, A.-W. (1998). *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen.* Springer.

[TAY11] Taylor, F. W., & Roesler, R. (2011). *Die Grundsätze wissenschaftlicher Betriebsführung.* Paderborn: Salzwasser Verlag GmbH.

[TIE06] Dipl.-Wirt.-Inf. Tiedemann, M. (2006). *Konzepte und Technologien zur Anwendungs-Integration.* Lübeck: Universität zu Lübeck.

[WAL11] Walls, Craig. (2011). *Spring in Action.* Shelter Island: Manning Publications Co.

[WBS13] Wikipedia -Business Software. (23.08.2013). *Wikipedia* . Von Wikipedia : en.wikipedia.org/wiki/Business_software abgerufen

[WFA13] wfApp2011-Activiti. (29.07.2013). *wfapp*. Von wfapp: de.wfapp2011.wikia.com/wiki/Activiti_1_(Tutorial) abgerufen

[WIJ13] Wikipedia - JEE. (14.08.2013). *Wikipedia*. Von Wikipedia: https:/de.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition abgerufen

[WIK13] Wikipedia - Komplexität. (26.08.2013). *Wikipedia*. Von Wikipedia: de.wikipedia.org/wiki/Komplexität abgerufeN

[WIN13] Wikipedia - Integration. (09.09.2013). *Wikipedia*. Von Wikipedia: de.wikipedia.org/wiki/Integration_(Software) abgerufen

[WIR13] Wikipedia-Rest. (29.07.2013). *Wikipedia*. Von Wikipedia: de.wikipedia.org/wiki/Representational_State_Transfer abgerufen

[WKO13] Wikipedia - Kopplung. (09.09.2013). *Wikipedia*. Von Wikipedia: de.wikipedia.org/wiki/Kopplung_(Softwareentwicklung) abgerufen

[WOL10] Wolff, E. (2010). *Spring 3: Framework für die Java-Entwicklung.* Heidelberg: dpunk.Verlag GmbH.

[WRM13] WfMC - Refernzmodell. (16.08.2013). *WfMC*. Von WfMC: http://www.wfmc.org/reference-model.html abgerufen

[WUU13] WfMC-ÜberUns (10.Juli.2013). *WfMC*. Von WfMC: http://www.wfmc.org/about-us.html abgerufen

[ZAH13] Zahn, C. (2013). *Praktikumsbericht - Workflow und Business Process Management Plattformen im Kfz-Leasing.* Mittweida.

Anhang

Anhang A - Postkorbmodul des Leasman

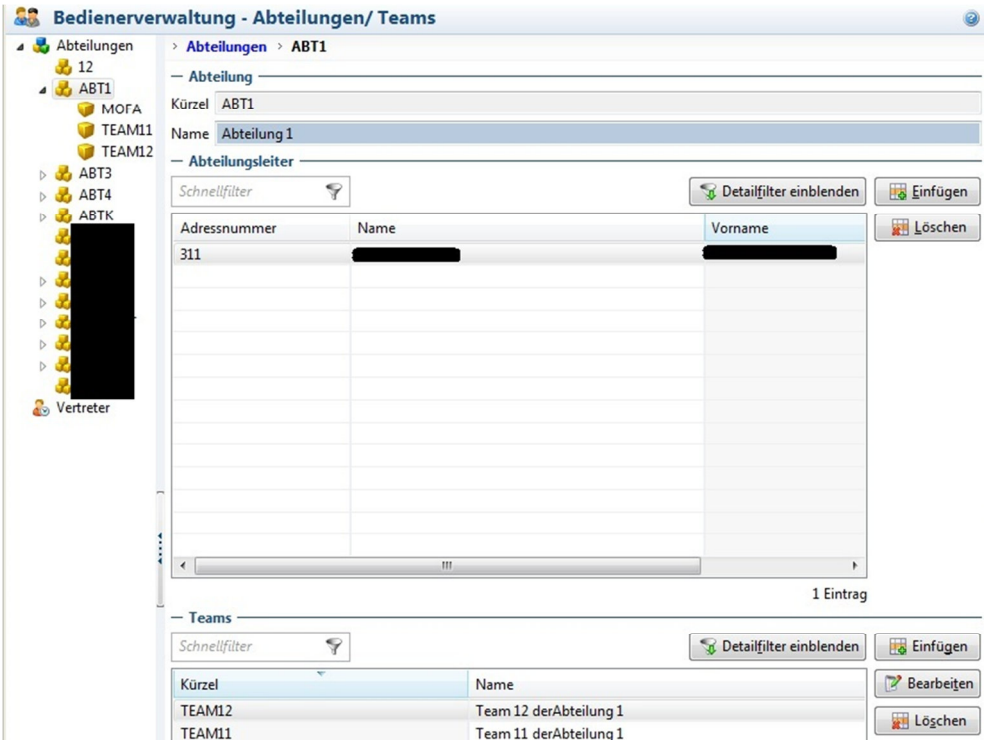


Abbildung 24: Abteilungs- und Teamoberfläche des Leasman

a) die Abteilungsübersicht des Leasman

b) die Postkorbübersicht des Leasman

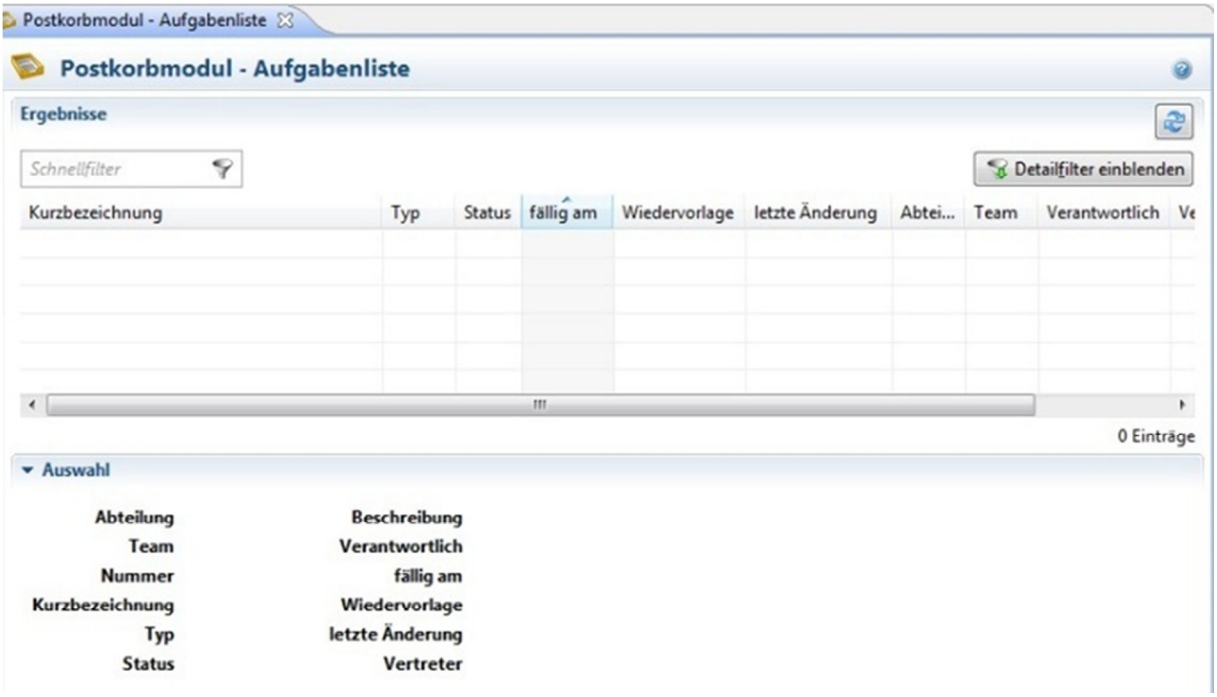


Abbildung 25: Das Postkorbmodul

Anhang B - Zeitlinie der BPM-Standards

Entwicklung von BPM-Sprachen

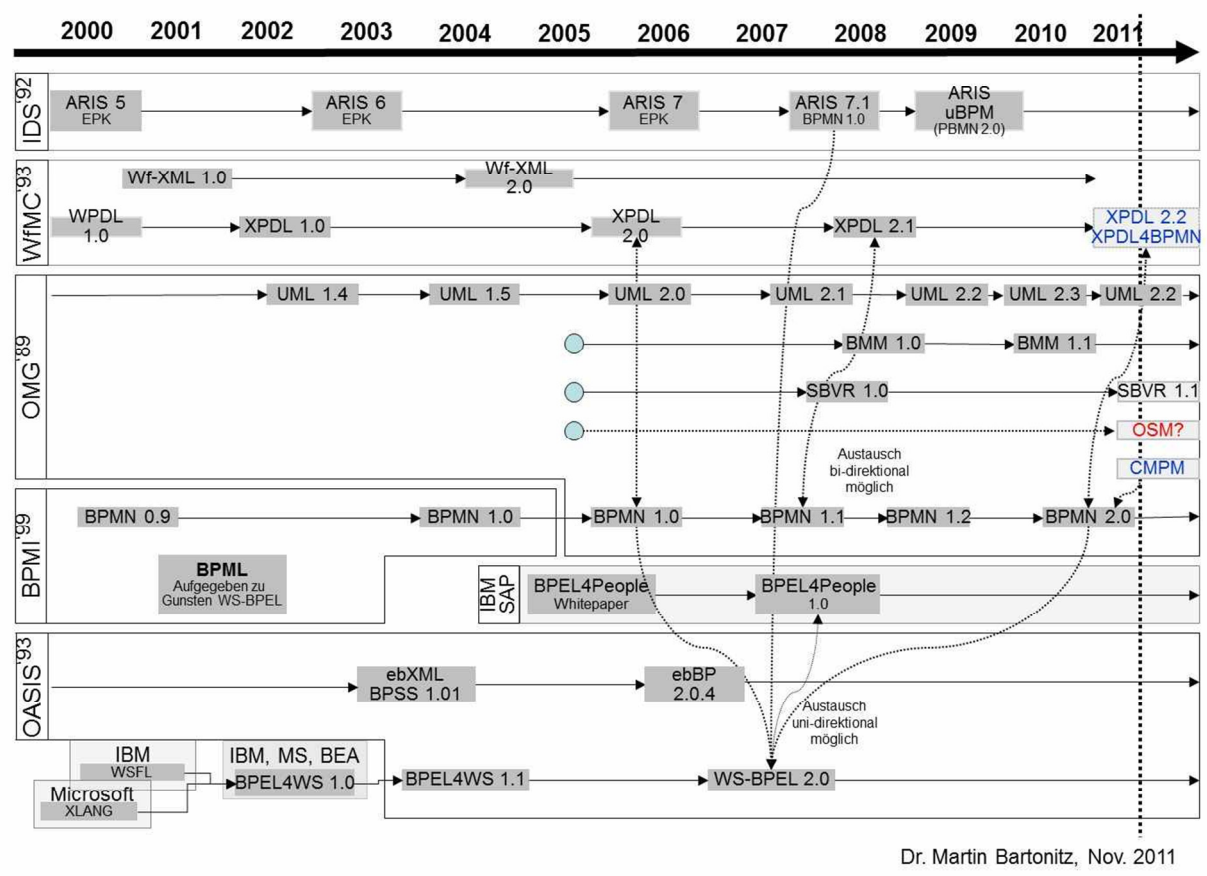
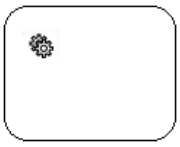
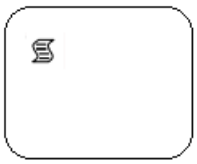
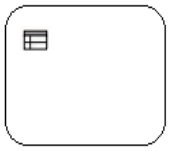
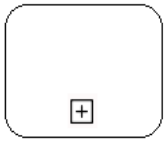
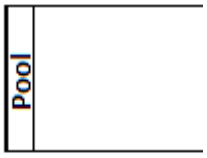


Abbildung 26: Entwicklung BPM-Standards[64]

[64] [BAR13] (Bartonitz, 05.07.2013 14:53)

Anhang C - Elemente Business Process Modeling and Notation

Übersicht ausgewählter BPMN-Elemente.[65]

Event	Symbol	Event	Symbol
Start Event		User Task	
Nachrichten Start Event		Service Task	
Timer Start Event		Script Task	
End Event		Business-Rule Task	
Nachrichten End Event		Subtasks	
Error End Event		Aufruf Aktivität	
Timer Boundary Event		Lane	
Kompensation Boundary Event		Pool	
Nachrichten Intermediate Event		Exklusiv Gateway	

[65] [POS13] (Berliner BPM-Offensive - BPMN Poster, 27.09.2013 – 10:36)

Signal-Intermediate Event		Parallel Gateway	
Timer-Intermediate Event		Ereignis-basiertes Gateway	
Allgemeiner Sequenzfluss		Inklusiv Gateway	
Bedingter Fluss		Nachrichten Fluss	
Standardfluss			

Tabelle 3: Elemente des BPMN

Anhang D - Funktionalitäten Business Process Management System

Grafische Übersicht über die Funktionalität eines BPMS

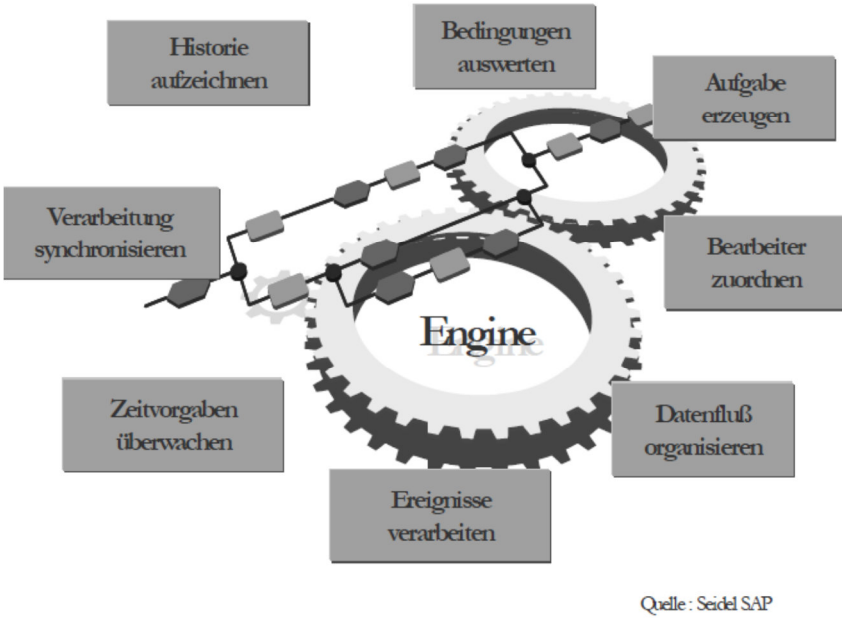


Abbildung 27: Funktionalität eines BPMS[66]

[66] [RIG09] (Riggert, ECM Enterprise Content Management, 2009, S. 79)

Anhang E - Activiti Modeler

Oberfläche Activiti Modeler und Signavio Process Editor

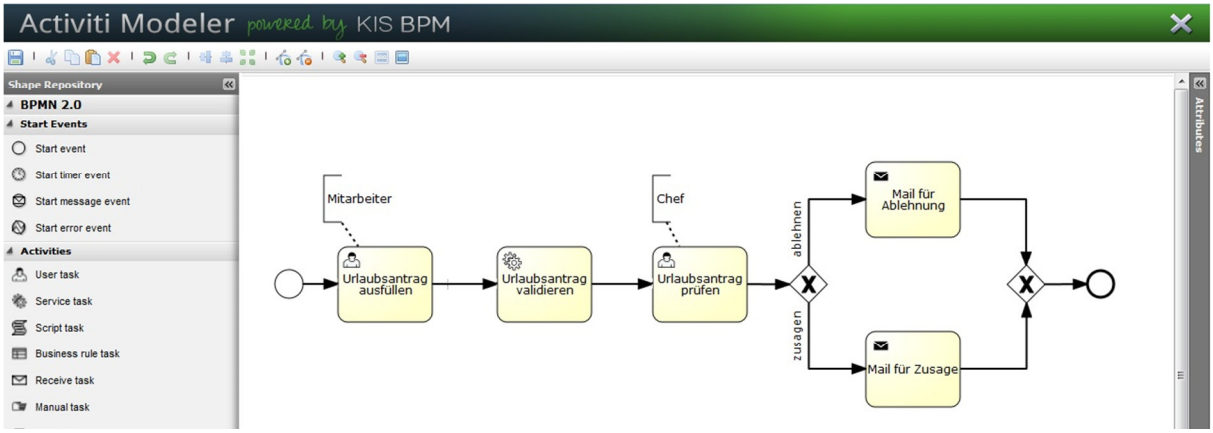


Abbildung 28: Oberfläche Activiti Modeler

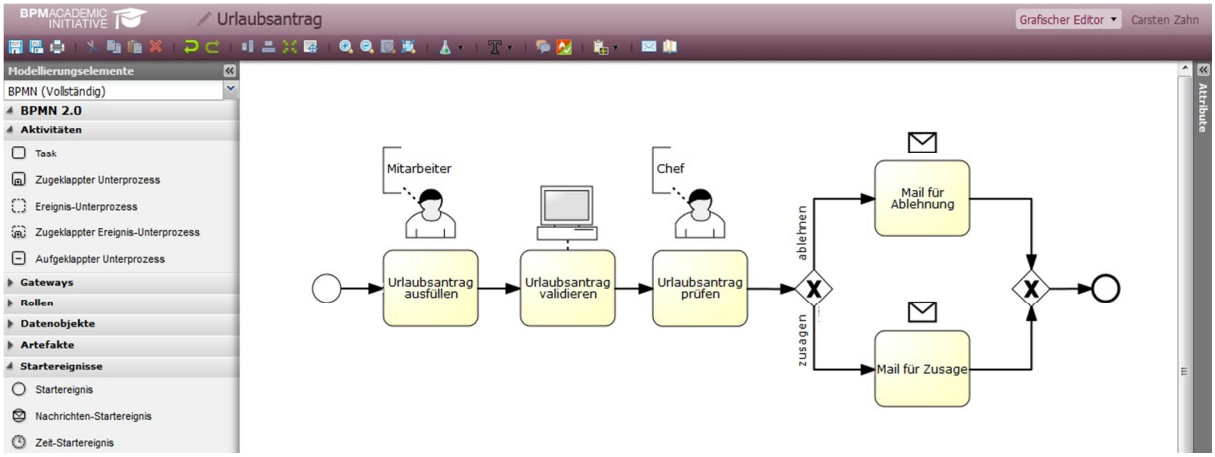


Abbildung 29: Oberfläche Signavio Process Editor

Anhang F- Repository

das Repository – Pattern nach [MSD13]

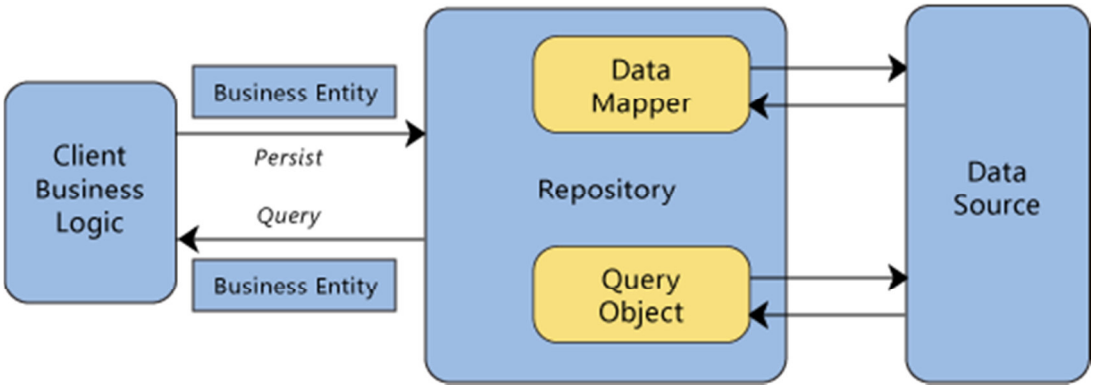


Abbildung 30: Repository Architektur[67]

[67] [MSD13] (MSDN Microsoft - The Repository Pattern, 16. 08 2013 – 15:13)

Anhang G - Beandefinitionen Activiti

1. die Datenbank-Bean für die Konfigurations-XML-Datei

```
<!--Standard Einstellungen -->
… (siehe Kapitel 6.2.2)
<!—zusätzliche Einstellungen mit ihren Standardwerten-->
        <property name =“jdbcMaxActiveConnections“ value=“10“>
        <property name =“jdbcMaxIdleConnections“ value=“10“>
        <property name =“jdbcMaxCheckoutTime“ value=“20000“>
        <property name =“jdbcMaxWaitTime“ value=“20000“>
```

a) **jdbcMaxActiveConnections**: maximale Anzahl an aktiven Verbindungen

b) **jdbcMaxIdleConnections**: maximale Anzahl an untätigen Verbindungen

c) **jdbcMaxCheckoutTime**: Zeit in Millisekunden bevor eine Verbindung geschlossen wird, nach der Zeit erfolgt eine gewaltsame Schließung

d) **jdbcMaxWaitTime**: Zeit in Millisekunden eine Verbindung wieder aufzubauen sowie die Erstellung eines Logstatus

e) **Tabelle der Datenbanken die Activiti unterstützt[68]:**

Datenbank	**Getestete Version**
H2	1.3.168
MySQL	5.1.21
Oracle	11.2.0.1.0
PostgreSQL	8.1
DB2	DB2 10.1
MS SQL Server	2008

Tabelle 4: Datenbankenübersicht Activiti

[68] [AUG13] (Activiti - User Guide, 02.08.2013 – 9:23)

```
<bean id="example.processEngineConfiguration" class="{Standardklassen frei wählbar, siehe
Standardklassen}">
        <!-- benötigten Properties -->
        …
<!-- zusätzliche Properties – vollständige Liste in der Klasse
                org.activiti.engine.iimpl.cfg.ProcessEngineConfigurationImple-->
        <property name =“databaseType“ value=“…“>
        <property name =“databaseSchemaUpdate“ value=“…“/>
        <property name =“jobExecutorActivate“ value=“…“/>
        <property name =“deploymentResources“ value=“…“/>
        <property name =“mailServerPort“ value=“…“/>
        <property name =“history“ value=“…“/>
```

2. Prozess Engine Konfiguration Bean

- **databaseType**

Muss benutzt werden wenn eine andere Datenbank als die h2-DB benutzt wird.

- **databaseSchemaUpdate:**

Gibt an wie die Prozess-Engine beim Starten und Beenden behandelt werden soll:

1. false: überprüft nur die Version des DB-Schemas (Library) mit dem der Datenbank (Standardwert)
2. true: Überprüft die Version der DB und führt notfalls ein Update durch oder erstellt diese
3. create-drop: Beim Starten der Engine wird die DB erstellt und beim Beenden geschlossen

- **jobExecutorActivate:**

Aktiviert oder deaktiviert den JobExecuter

1. true: der JobExecuter führt seine Funktionen, wie das rechtzeitige Ausgeben von z.B. Intermediate Events, aus
2. false: der JobExecuter wird deaktiviert

- **deploymentResources:**

setzt beim Hochfahren der Engine automatisch den angegebenen Prozess ein

- **mailServerPort:**

Wird benötigt wenn der Mail-Server-Port nicht der standardisierte Port ist.

- **history:**

Gibt an wie viele Daten des Prozesses gespeichert werden:

1. none - speichert keinerlei Informationen
2. activity - speichert nur die Prozess- und Aktivitätsinstanzen sowie den Abschlusswerte des Prozesses
3. audit - speichert die Prozess- und Aktivitätsinstanzen sowie ununterbrochen die Variablen (Standardwert)
4. full – speichert alle Informationen die innerhalb des Prozesses entstehen

- **Standardklassen für die Einstellungen einer Engine**

Von Activiti vorgegebene Klassen mit Einstellungen für die Prozess Engine:

1. **org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration**:
   - wird bei Standalone-Konfiguration benutzt
   - Activiti gibt auf die Transaktionen acht
2. **org.activiti.engine.impl.cfg.StandaloneInMemProcessEngineConfiguration**:
   - Activiti sorgt für die Transaktionen
   - es wird eine h2 in-memory DB genutzt
   - die DB wird beim Starten der Engine erstellt und beim Beenden geschlossen
3. **org.activiti.spring.SpringProcessEngineConfiguration**:
   - wird genutzt, wenn die Engine in einer Springumgebung eingesetzt wird
4. **org.activiti.engine.impl.cfg.JtaProcessEngineConfiguration**:
   - wird bei einer Standalone-Konfiguration genutzt, wenn die Transaktionen über eine JTA läuft

Anhang H – UML-Diagrammtypen zur Vertragsanbahnung

1. Aktivitätsdiagramm

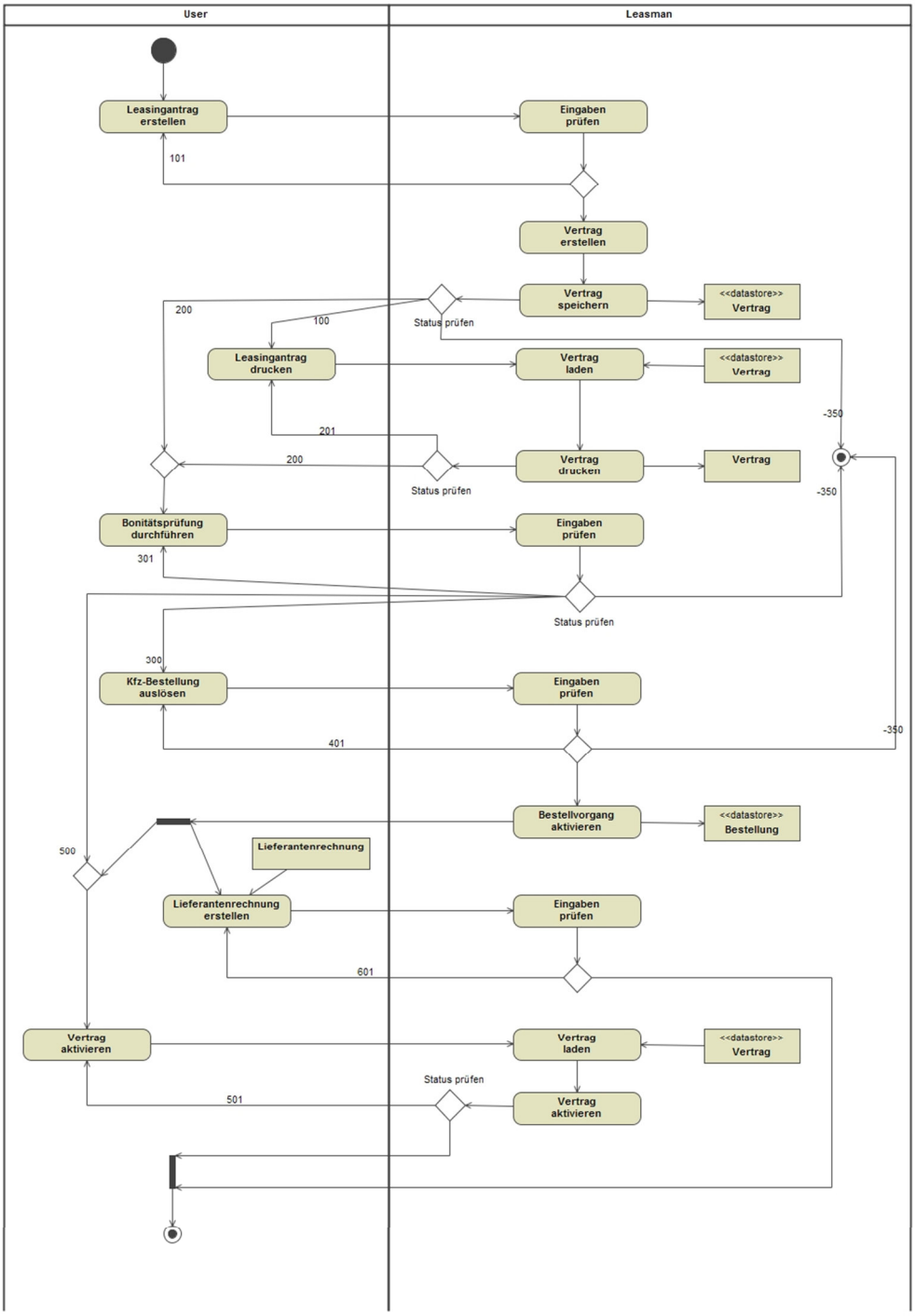


Abbildung 31: Aktivitätsdiagramm Vertragsanbahnung

2. Prozessdiagramm

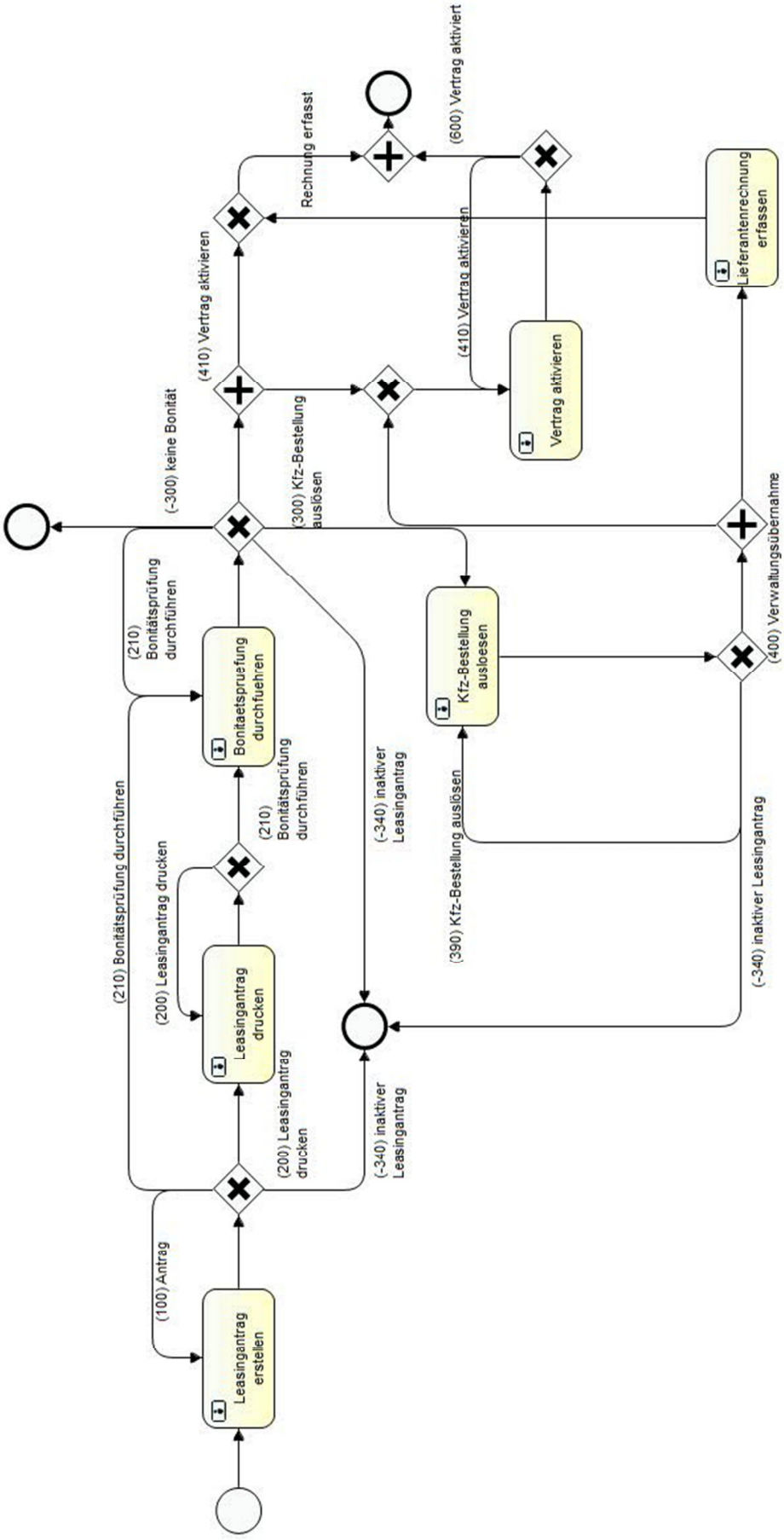


Abbildung 32: Prozessdiagramm Vertragsanbahnung

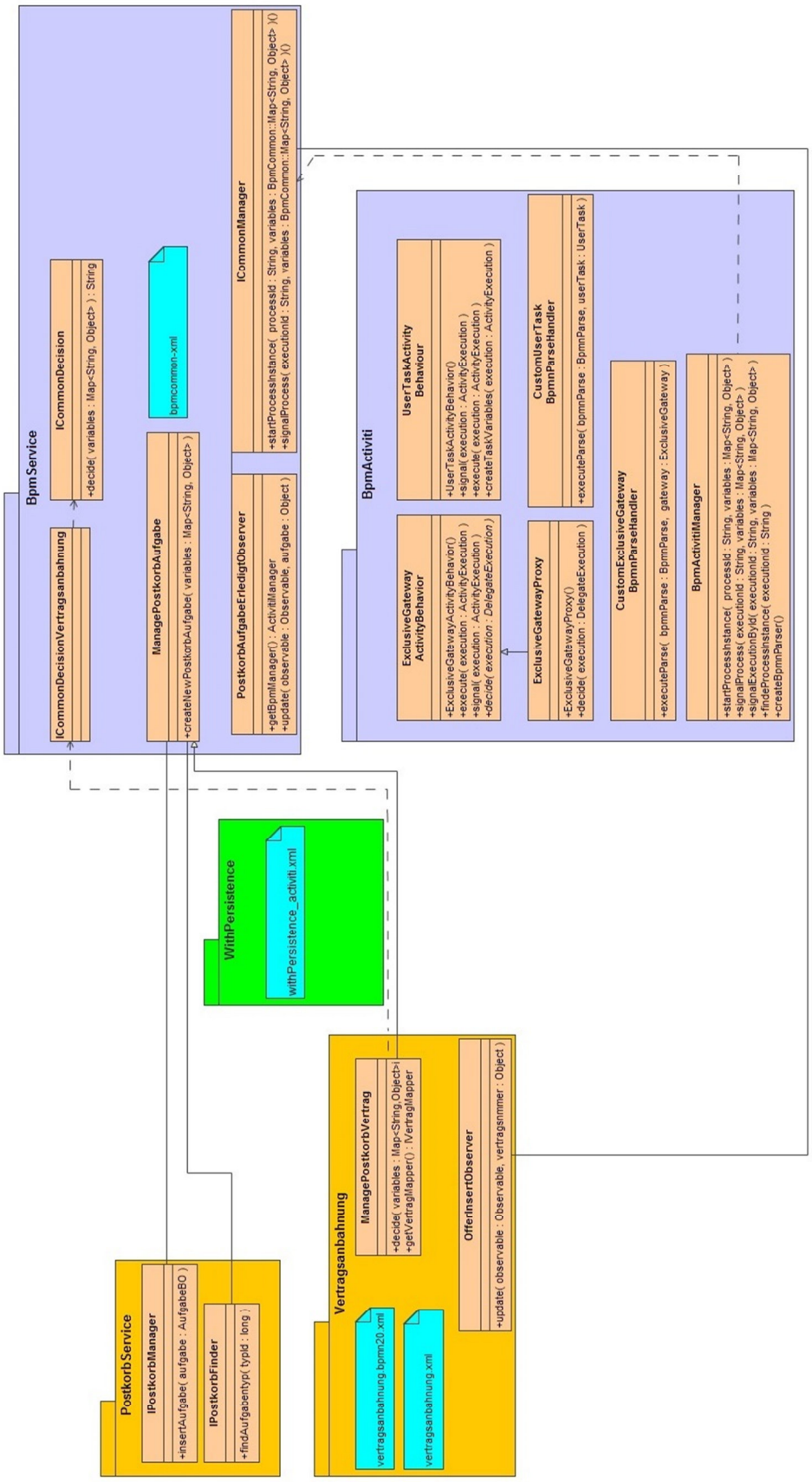


Abbildung 33: Klassendiagramm Vertragsanbahnung

3. Klassendiagramm

Anhang I – Listener-Reihenfolge

1. Reihenfolge der Execution- und Task-Listener-Events

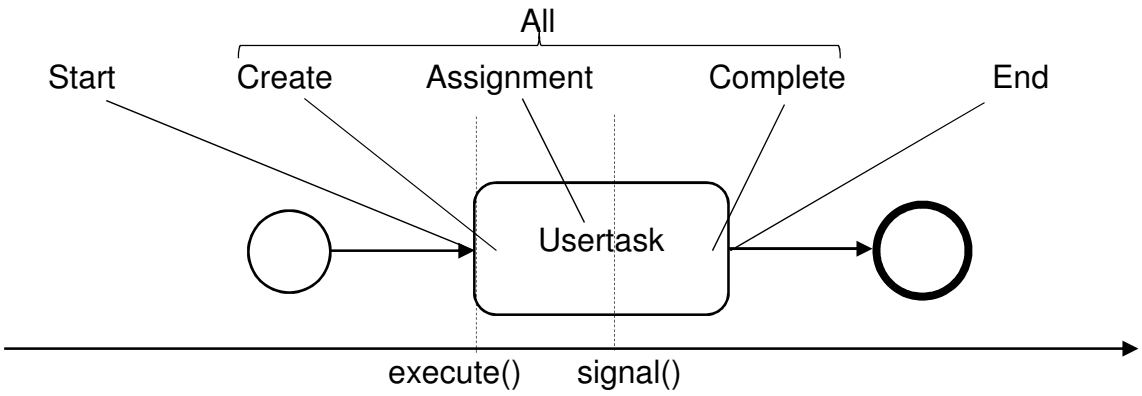


Abbildung 34: Reihenfolge des Listeneraufrufs

Anhang J - Ausschnitt der Leasman - Datenbankeinträge

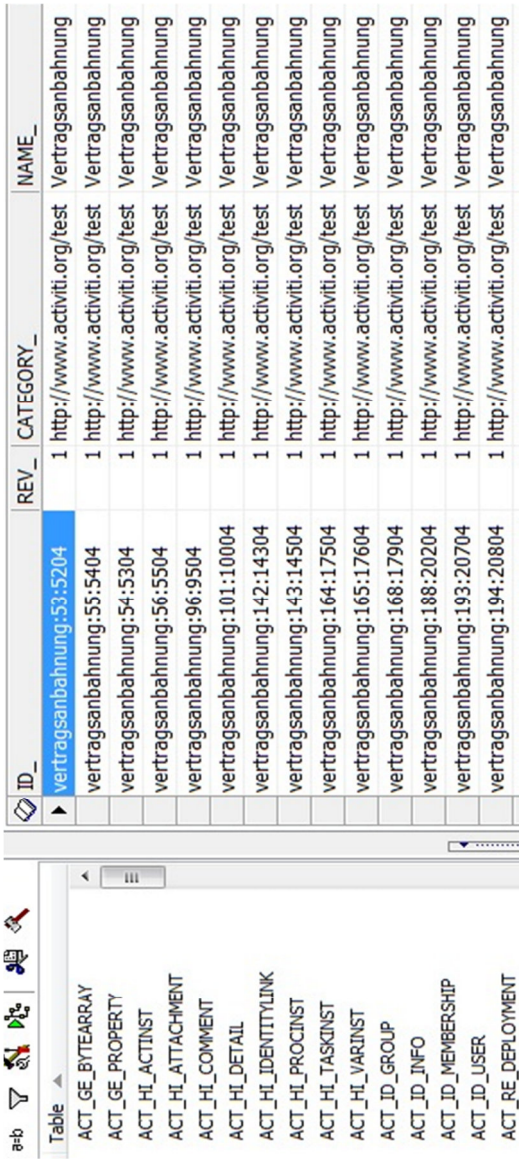


Abbildung 35: Datenbankeinträge von Activiti

Abbildung eines Leasman-Datenbankeintrags von Activiti

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Thema „Integration einer BPM-Engine in eine komplexe Geschäftsanwendung“ eigenständig durchgeführt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden´, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Ich bin mir darüber bewusst, dass eine diesbezügliche Falschangabe rechtliche Schritte nach sich ziehen wird.

Mittweida, 11.10.2013

_____

Carsten Zahn